Robust Randomized Trajectory Planning for Satellite Attitude Tracking Control

by

Drew Richard Barker, ENS (USN)

B.S. Physics
United States Naval Academy, 2004

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

Author .....
Department of Aeronautics and Astronautics
May 12, 2006

Certified by .....
Leena Singh
Charles Stark Draper Laboratory
Thesis Supervisor

Certified by .....
Jonathan How
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Jaime Peraire
Professor of Aeronautics and Astronautics
Chairman, Department Committee on Graduate Students

Robust Randomized Trajectory Planning for Satellite Attitude Tracking Control

by

Drew Richard Barker, ENS (USN)

Submitted to the Department of Aeronautics and Astronautics
on May 12, 2006, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

This thesis presents a novel guidance strategy that uses a randomized trajectory planning algorithm in a closed-loop fashion to provide robust motion planning and execution. By closing the guidance, navigation, and control loop around a randomized trajectory planning algorithm, a robotic vehicle can autonomously maneuver through a field of moving obstacles in a robust manner. The guidance strategy provides executable plans that are robust to known error sources when supplied with an estimate of the initial state, the goal, the predicted locations of obstacles, and bounds on error sources affecting the execution of a planned trajectory. The planning function presented in this thesis extends the Rapidly-exploring Random Tree algorithm to dynamic environments by exploring the configuration-$\times$-time space using a node selection metric based on the maneuvering capability of the vehicle. The guidance strategy and the new randomized trajectory planning algorithm are applied to a challenging satellite attitude guidance problem in simulation.

Thesis Supervisor: Leena Singh
Title: Charles Stark Draper Laboratory

Thesis Supervisor: Jonathan How
Title: Professor of Aeronautics and Astronautics

Acknowledgments

Thanks to the Creator, who in splendor furnished the universe with marvelous things great and small which are all worthy of study, enjoyment, and appreciation. For my parents and my brother, their endless love and support continues to be a pillar of strength for me; and for all those in whose love I live.

A special thanks to those here at MIT that have helped me along the way, particularly my advisors Leena Singh and Jon How. Thanks also to Ernie Griffith, for his patients and interest in my ultimate understanding, and the rest of the team I worked alongside at Draper Labs: Len Epstein, Karl Flueckiger, and Ed Bergmann. Finally, thanks also to Stephen Thrasher, Jon Beaton, Brian Mihok, Tom Krenzke and the other Draper Lab Fellows with whom I shared the experience.

Permission is hereby granted by the author to the Massachusetts Institute of Technology to reproduce any or all of this thesis.

[This page intentionally left blank].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This research involves the study of a novel satellite attitude guidance strategy that uses a randomized trajectory planning algorithm in a closed-loop fashion to provide robust motion planning and execution. The main contributions of this thesis include a method for providing robust, receding-horizon motion planning with the use of an open-loop planning algorithm, and a new randomized trajectory planning algorithm that extends the Rapidly-exploring Random Tree algorithm [33] to dynamic environments. The methodology used to formulate a robust plan with an open-loop planner will be referred to as the Robust, Receding-Horizon Concept (RRHC), while the new planning algorithm will be referred to as the Randomized Spacetime Trajectory Planner (RSTP). The new planner is incorporated into the robust planning methodology and applied to a challenging satellite attitude guidance problem. The guidance problem is defined by an experiment which aims to demonstrate that two microsatellites in low earth orbit can provide a refined estimate of an exoatmospheric object's position and velocity by tracking the object with onboard cameras. The methodology and the planning algorithm are designed to be scalable to larger configurations.

The experiment used to define a challenging satellite attitude guidance problem is referred to as the Orbital Tracking Experiment (OTE). The OTE is an experiment invented expressly for testing the RRHC and the RSTP in a simulation context. General satellite attitude guidance requirements are derived from the experimental objectives of the OTE. The derived requirements are used to decide upon the hardware and sensors which are modeled in the simulation test bed. The hardware and sensors define specific constraints and allow for a specific problem statement to be defined. The RRHC and the RSTP are then applied to solve the attitude guidance problem in simulation.

The satellites of OTE are tasked with tracking an exoatmospheric object (another satellite, space debris, etc.) in order to provide an estimate of the object's position and velocity, which is otherwise known as the object's translational state. The translational state estimate of the object can be derived from line-of-sight measurements supplied by cameras mounted on each of the two satellites. The translational state of the object is not actually estimated within the simulations because the OTE setup is only used as a proving ground for the attitude guidance algorithms. In order to take line-of-sight measurements, the satellite must slew from an initial orientation to

an orientation where the object image resides within the field of view of an onboard camera, which is referred to as the primary camera. The attitude guidance problem defined by the pre-acquisition phase of the OTE (orienting the satellite to obtain the object) is significantly different from the attitude guidance problem defined by the post-acquisition phase of the OTE (orienting the satellite to track the object). The research presented in this thesis focuses on solving the attitude guidance problem defined by the post-acquisition phase of the OTE.

1.1 General Requirements

The overall objective of the OTE is to provide a translational state estimate of an exoatmospheric object. The main objectives for one of the two satellites supporting OTE during the post-acquisition phase is to provide line-of-sight measurements of the object, while also maintaining accurate estimates of own-ship position, velocity, and orientation with respect to an inertial frame. Furthermore, each individual satellite must also communicate its own measurements with the other in order to calculate a three-dimensional translational state estimate of the tracked object. Simply stated, the requirements placed upon the satellite attitude guidance system are to orient the satellite so that the objectives are best accomplished. Therefore, the satellite attitude guidance system must maintain the object within the field of view of the primary camera while also orienting the satellite to permit both inter-satellite communications and instrument sensing of the satellite's inertial position and orientation.

1.2 Hardware and Sensors

The following hardware and sensors are selected to be simulated based upon the mission objectives of the OTE. A primary camera is selected for the purposes of tracking the object. A GPS receiver supplies own-ship inertial position and velocity sensing. An Inertial Measurement Unit or IMU provides angular velocity estimates, and a star camera supplies attitude estimates. The angular velocity estimates from the IMU and the attitude estimates from the star camera are optimally combined in a Kalman filter to produce the estimated rotational state of the satellite. Additionally, reaction wheels supply the torque necessary to change the orientation of the satellite. Finally, a communications antenna is selected to relay the measurement data to the other satellite. Naturally, other hardware items are required to operate a satellite; however, since they do not directly impact the attitude guidance problem they have been omitted from the simulation.

1.3 Problem Constraints

The sensor requirements and mission objectives are both reflected in the definition of the constraints placed upon the attitude guidance system. To fulfill the mission objectives, the primary camera must point at the object at all times. This objective

is formulated as an attitude constraint in order to force a solution that points the primary camera at the object at all times, provided such a solution exists. The pointing constraint requiring the object to remain in the field of view of the primary camera is an inclusion cone constraint. That is to say that the bore-sight of the primary camera must remain inside of a cone that is centered on the line-of-sight vector from the satellite to the object and is as wide as the field of view of the primary camera. The star camera adds additional constraints to the attitude guidance problem because the star camera can only update the rotational state estimate while it is pointed at stars. In other words, the bore-sight of the star camera cannot point within cones which are defined by the line-of-sight vectors from the satellite to the Sun, Earth, and Moon, by the size of each obstruction, and by the field of view of the star camera. The inclusion or exclusion cone constraints are defined in the simulation by the line-of-sight vector and a cone half angle, α. For an illustration of the solar exclusion cone see Figure 1-1. The communications antenna must maintain a certain orientation in order to transmit and receive information from the other satellite. The sensitivity band of an antenna, the amount of power devoted to communications, and the distance between the satellites dictates the size of the exclusion cone constraint placed on the direction that the antenna is pointed relative to the other satellite. It is assumed that the GPS antennae placed on the satellite are sensitive to signals coming from any direction, so they do not induce an additional constraint on the attitude guidance problem. No thermal or solar array constraints will be factored into the tracking phase requirements.

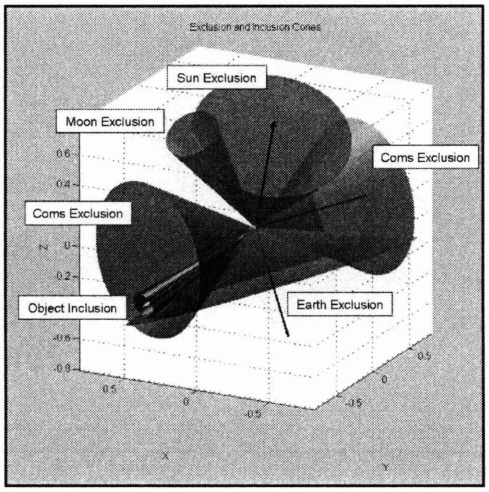

Figure 1-1: The solar exclusion cone constraint, defined by the line-of-sight vector from the satellite to the sun and the exclusion cone half-angle, α.

It is important to note that all of the pointing constraints that the satellites must negotiate are predictable. The ephemerides of the Sun, Earth, and Moon are well known, each satellite is expected to have the ability to monitor and predict its own location, and if the tracked object does not fire thrust jets, then its position can be predicted by using the astrodynamic equations of motion. From the ephemerides and

predicted locations of the satellites and the object, all pointing constraints can be derived for future times.

Besides the pointing constraints placed on the satellite's orientation, there are other constraints placed upon the rotational motion of the satellite which stem from the satellite's dynamic capability. Specifically, the reaction control wheels manipulation of the attitude of the satellite is subject to limited angular velocity and limited angular acceleration.

1.4 Problem Statement and Approach

Consider two basic approaches to the attitude guidance problem presented; a reactive approach and a planning approach. A reactive approach maneuvers the satellite based solely upon tracking errors and the location of constraint boundaries. A planning approach maneuvers the satellite based on the results of a planning function that incorporates the future location of the pointing constraints and the vehicle's dynamic capability into a plan designed to successfully navigate the constraint boundaries. Because the inclusion and exclusion constraint cones mentioned in the previous section can move and overlap, a purely reactive guidance algorithm cannot offer any guarantee of success. See Figure 1-2 for an example of how a purely reactive algorithm could easily become trapped behind obstacles.

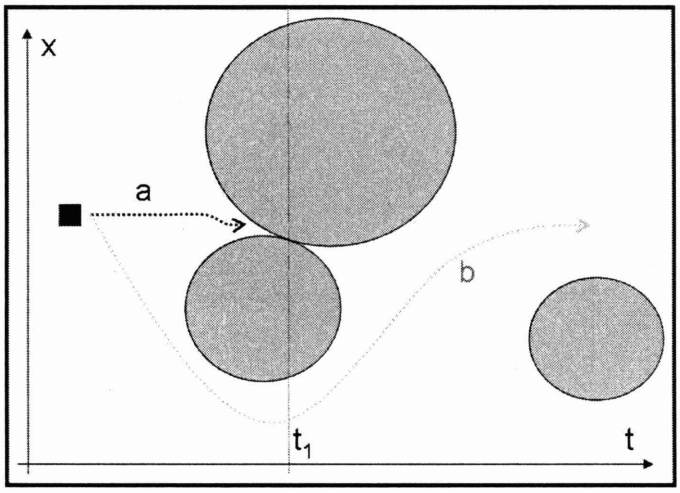

Figure 1-2: A simple vehicle, depicted as a black box, can change its position along the x axis in time. The shaded circles represent obstacles that occupy a portion of the x axis in time. A purely reactive guidance strategy would take path (a) and fail when two obstacles form a wall at $t = t_1$. If the obstacles are predictable and the guidance strategy can generate a plan, then a path like (b) can be found so the obstacles are avoided.

There are a number of existing planning algorithms that are capable of producing a solution that successfully navigates the pointing constraints while remaining

within the limits of the satellite's capability. Optimal control techniques, dynamic programming, and potential field methods are a few starting points for planning approaches that might be explored for single satellite attitude guidance. However, if the problem is scaled to a large configuration with a high dimensional state space, the aforementioned techniques break down when a timely solution is required. Fortunately, a considerable amount of research has been performed in recent years on problems where a plan must be quickly formulated for a problem with high dimensional state spaces, kinematic and dynamic constraints, and moving obstacles. The most widely recognized algorithms that are designed to produce open-loop solutions for such problems are described as randomized trajectory planners, and a few are presented in References [17] [34] and [22].

However, the planning problem associated with the OTE attitude guidance requirements is larger than simply developing an open-loop motion plan. The attitude guidance system must not only produce a motion plan for the satellite, but it must produce a robust motion plan that accounts for the fact that the motion plan is based on sensor measurements and state estimates that are error prone, and that those errors impact the formulation of the plan. The problem statement capturing the attitude guidance problem approached in this thesis is summarized as follows: *Formulate a guidance strategy that generates a timely motion plan for a high dimensional problem that is robust to system errors and uncertainties and involves kinematic constraints, dynamic constraints, and predictable moving obstacles.* The remainder of this thesis describes the application of a new randomized trajectory planning algorithm and a means of incorporating an open-loop planning algorithm into a robust receding horizon planning strategy that solves the problem just presented.

1.5 Organization of Thesis

Following on the introduction, Chapter 2 presents background information on motion planning and describes the framework used to generate a robust motion plan for satellite attitude control through the employment of a randomized trajectory planner. Chapter 3 presents the components of the inner loop of the simulation, which consists of a standard feedback control system. The navigation function used in the experiment operates on a feed-back implementation of a Kalman filter that provides estimates of attitude error states. Kalman filter error states are used to correct the rotational state estimate, which is used by the inner-loop feedback controller. Future translational states of the satellite and object are also predicted by the navigation system and directed to the outer-loop for the formulation of the motion plan. A simple tracking control law is also developed in Chapter 3 which helps simplify the planning process. Chapter 4 discusses the outer loop, which is where the motion plan is generated. First, a pre-planning function is introduced. The pre-planning function is critical to producing a robust plan and uses estimates of the upper bounds of various errors to return information necessary for generating a robust plan. The Randomized Spacetime Trajectory planner is also introduced in Chapter 4, and the specific application of the algorithm to the OTE is described. Experimental results

and subsequent discussion are found in Chapter 5 as well as ideas for future work. Finally, Appendix A provides a reference table specifying the notation and convention used throughout the paper.

Chapter 2

Background and Framework

Recent advances in the areas of computer science and control systems has catalyzed research efforts aimed at developing robotic systems that are increasingly more autonomous. A major focus of autonomous systems research is centered on developing reliable and efficient motion planning algorithms. Motion planning applies to a very broad spectrum of problems that have applications in everything from mobile robots, manipulators, and spacecraft, to digital animation and the analysis of the configuration spaces of flexible molecules [33, 30]. In essence, the solution of a motion planning problem is an executable plan that moves a system from one state or configuration to another in a manner that avoids collisions with obstacles and preferably minimizes some a function associated with a measure of the system's performance.

The satellites of the OTE are intended to be autonomous in the sense that the network of satellites is provided with general instructions on what to do (track a moving object) instead of specific commands on how to accomplish the task. Therefore, the satellites must be able to generate high level plans and execute those plans with lower level functions. In the case where a network of satellites is tasked to track multiple objects simultaneously, an autonomous decision would have to be made as to which satellites will track which objects. Each individual satellite must autonomously decide how it will orient itself so that it can successfully track the object while maintaining communications with the other satellites, preventing the star camera from pointing at the Sun, Earth, or Moon, and satisfying any other constraints placed on the system. For the individual satellite, this reduces to a motion planning problem.

The real world implementation of a motion plan often requires nonlinear, dynamic systems to execute a plan in real time within environments that are both uncertain and dynamic. Some motion planning problems involve systems and/or environments that can be accurately represented with simple models that only have a few degrees of freedom. These simple low dimensional problems can often be solved by algorithms capable of finding optimal solutions where solutions exist. Other motion planning problems involve systems with many degrees of freedom and cannot be represented in simplistic models. For many of these complex, high dimensional problems, optimal solutions are often computationally intractable and assurances on the quality of the solution must be compromised in order to find a feasible solution in a timely fashion. Whether the problem is simple or complex, there are a number of concepts that are

common to most motion planning problems. A brief summary of some of these basic concepts follows.

2.1 Motion Planning Definitions

State space: The state of a system consists of a set of parameters that uniquely captures the nature of the system with respect to the problem to be solved. Consider how a set of parameters may uniquely define the state of a system, vehicle, robot or object within the context of its environment. For example, within the context of a translational motion path planning problem, the state of a point mass may be adequately described by its position and velocity. Similarly, within the context of a rotational motion path planning problem, an object's state may be defined by the Euler angles that describe the objects orientation and a set of angular velocities. The state space is the set of all possible states for the system in its environment.

Configuration space: The notion of a configuration space is a useful tool for motion planning. The configuration space is a subset of the state space that is often represented as a collection of states that share a common constant form or function. For example, a configuration of a robot or vehicle could be its position with respect to an inertial reference frame, or it could be a trim trajectory where velocities are held constant by a constant control input. The configuration space is the set of all possible configurations of the robot or vehicle.

Free space: The free space is a subset of the state space that is free of any obstacle. Any successfully planned path will lie within the free space. Note that the free space does not account for all the constraints that may exist with a motion planning problem. The nature of nonholonomic and dynamic constraints (see below) prevents them from being represented as subsets of the state or configuration spaces.

Maneuver space and trim trajectory: A trim trajectory is often characterized by the condition where a subset of the state parameters remains constant with the use of constant control inputs. Maneuvers are defined as realizable transitions between trim trajectories. The maneuver space is the set of all maneuvers that a system is capable of executing. Decomposing a vehicle's motion into trim trajectories and maneuvers is useful for planning the motion of an underactuated vehicle. For example, when planning the motion of a glider, it is easier to formulate the planning problem in terms of trim trajectories and maneuvers because the glider is capable of maintaining a trim trajectory for an extended period of time while it is not capable of remaining in an inertial position for

more than an instant.

Feasibility: In the context of motion planning problems, a problem is feasible if a motion plan exists that will bring the system from an initial state to a desired state without violating any of the constraints posed in the problem.

Kinematic constraints: Constraints derived from the mechanics of motion without regard to the forces that cause the motion. For example, if a wheel rolls without slipping, the velocity of the center of mass of the wheel is kinematically constrained to be equal to the cross product of the velocity with a vector from the point of contact with the surface to the center of mass of the wheel.

Dynamic constraints: Constraints that are the result of the mechanics of the system due to forces applied to the system. Dynamic constraints capture the reason why a vehicle cannot instantaneously go from a stop to a high velocity. The acceleration of the vehicle is dynamically constrained to obey the equations of motion which account for the effect of forces applied to the vehicle.

Holonomic constraints: Integrable constraints that restrict the system to a hyper-surface of the configuration space. Holonomic constraints can be formulated as equality constraints between the parameters of the configuration space, thereby reducing the dimension of the configuration space. Consider a car with its front wheels fixed to some angle of turn. The vehicle is restricted to travel in a circle which can be considered a hyper-surface of the original configuration space.

Nonholonomic constraints: Nonintegrable constraints that do not restrict a system's configuration space. A nonholonomic constraint can be formulated as a nonintegrable equation involving configuration parameters and their derivatives. Consider how a car can only turn its front wheels. The car's instantaneous velocity is restricted in that the car cannot move laterally because the back wheels do not also turn. However, the configuration space of the car is not reduced because the car can obtain any configuration through a series of maneuvers.

External constraints: External constraints, which are sometimes referred to as obstacles, reside outside of the system and confine the state or the output space of the system. Obstacles can be static or moving and will occupy a portion of the configuration space at a given time.

Reachability: One point in the configuration space is reachable from another point if a dynamically feasible connected path exists between them. That is to say

that there exists a sequence of admissible controls that take the system from the initial point to the final point.

Computational complexity: Computational complexity is a measure of the computational effort required to generate a motion planning solution within the context of a defined motion planning problem. The authors of [42] and others have analyzed classes of motion planning problems to classify them in terms of their complexity. A problem determined to be class P hard is a problem that can be solved by an algorithm of polynomial time complexity on a deterministic Turing machine. A problem determined to be NP hard is a problem that can be solved by an algorithm of polynomial time complexity on a nondeterministic Turing machine; which is a machine that has the ability to perform an unlimited number of independent computational sequences in parallel. A PSPACE hard problem is one which can be solved by an algorithm of polynomial space complexity on a nondeterministic machine. Key parameters used to classify the computational complexity of a problem include the dimension of the problem and the nature of the constraints placed on the problem [31]. In a less formal manner, the length of time required by a particular computer using a particular algorithm to solve the problem can also be used as a measure of the algorithm's computational performance. For more information on computational complexity calculations see [31] [46] or [11]. Computational complexity can be thought of both in terms of lower bounds, where the nature of the problem demands a minimal amount of computation in order to solve it; and in terms of upper bounds, where a particular algorithm used to solve the problem uses a certain number of computations to arrive at a solution.

Completeness: If a feasible path from the initial to the final configuration exists, then a motion planning algorithm is called complete if it always able to find a feasible path. Furthermore, a complete motion plan will only return a failure if the problem is infeasible. Probabilistic motion planners attempt to find feasible solutions by incrementally searching the state space via random inputs or attempts to connect random goal configurations. Probabilistic motion planners are deemed probabilistically complete if the probability of finding an existing feasible solution approaches one as the number of iterations approaches infinity.

2.2 Motion Planning Catagories

Most motion planning algorithms fall into one of three general categories of motion planning methods. Each category is founded on an intuitive approach to generating a path through a field of obstacles. The categories include cell decomposition methods, artificial potential field methods, and roadmap methods.

A cell decomposition method partitions the free space into a finite number of connected regions or cells in which it is easy to connect any two configurations. This

takes a difficult problem and divides it into many smaller, simpler problems. The motion planning algorithm is tasked with breaking the configuration space down into cells, finding a set of neighboring cells that collectively contain both the initial configuration and the goal configuration, and then generating a motion plan in each of the cells within the solution set. This method often requires large computations to be made before starting to generate a motion plan in order to partition the configuration space. This approach is best suited for problems with static environments and low dimensional configuration spaces.

Where cell decomposition methods partition the free space around obstacles in order to avoid them, artificial potential field methods artificially assign obstacles repulsive forces to avoid them. The repulsive forces around the obstacles are typically defined by the negative gradient of a potential function that in addition to having steep slopes near obstacles contains a global minimum at the goal configuration. In this way, the vehicle is not only repelled by the obstacles, but it is also drawn towards the goal. Potential function methods have also been referred to as local methods because the motion of the vehicle is influenced by the local environment. This characteristic of potential field methods makes them ideal for real-time collision avoidance schemes. As obstacles in the local environment are sensed, the potential field is redefined to prevent the vehicle from colliding with the obstacle. The main drawback of this approach is that artificial potential field methods are susceptible to local minima where the vehicle can become stuck. In some circumstances a potential field can be defined that does not contain a local minimum. The resulting planner is said to operate on what is called a navigation function. Operating with a navigation function is an enticing motion planning strategy; however, computing a navigation function for a general case could be as difficult as solving the motion planning problem for all initial conditions [17]. Artificial potential field methods are often effective, but generally do not offer completeness guarantees and are not easily formulated for high dimensional problems.

Instead of formulating a single path from the initial configuration to the goal configuration, as is typically accomplished in cell decomposition an artificial potential field methods, roadmap methods attempt to capture the connectivity of the free space by generating a network of collision-free connecting paths called a roadmap. Roadmap methods can be used for multiple path-planning queries because the same network of paths is used to find a connection from any initial configuration to any goal within the same environment. Finding a motion planning solution using a roadmap method usually requires three general steps. First, the network of collision free connected paths in the free space must be generated. Second, the initial and goal configurations must be connected to the network. Finally, the proper sequence of connected paths must be chosen to move from the initial configuration to the goal. Roadmaps are typically generated in a preplanning stage and often make use of Voronoi diagrams, visibility graphs, and connectivity graphs.

2.3 Probabalistic Motion Planning

While algorithms representing each method just described arrive at a motion planning solution from a different approach, most suffer in terms of computational complexity when applied to problems that contain high dimensional configuration spaces. According to results presented in [31] and [16] the complexity of path planning tends to increase exponentially with the dimension of the configuration space, which would explain why many algorithms fail to yield timely solutions when applied to high dimensional problems. This curse of dimensionality has fueled research in a new category of motion planning methods that can often produce solutions quickly for high dimensional problems through the means of a random search of the configuration space. Algorithms within this new category, referred to as probabilistic methods, admittedly forgo optimal solutions for the quick discovery of a feasible solution. The probabilistic methods that have emerged within the last decade have experienced great popularity, and their application will be further explored in this thesis.

2.3.1 Probabalistic Road Maps

One of the earliest Probabilistic methods was developed by Kavraki *et al.* and termed Probabilistic Roadmaps (PRM). To generate the network of collision free connected paths used in a roadmap planner, a PRM chooses a number of uniformly sampled random configurations and attempts to connect each of the sampled configurations with a simple connection strategy. The PRM is generated before the actual path planning problem is posed based on knowledge of obstacles in the environment. Attempts are made to connect the initial and final configurations to the pre-computed roadmap via the same connection strategy used to generate the PRM. Once those connections are made, the correct sequence of nodes must be chosen to move from the initial configuration to the goal configuration [28]. The PRM was developed to reduce the computational complexity of generating motion paths for robots with more than 4 degrees of freedom in realistic environments [29]. Like traditional roadmap methods, the PRM is based on generating a roadmap that can be accessed for multiple path planning quarries through the same environment quickly and efficiently. The randomly selected configurations used to generate the roadmap are referred to as milestones, and it has been shown that as the number of milestones increases, the probability of successfully finding a correct solution where one exists approaches unity. The PRM is limited by the fact that it does not account for dynamic environments[1], the pre-computed roadmap is not easily regenerated on the fly, and the PRM does not take the dynamics of the vehicle into account.

2.3.2 Rapidly-exploring Random Tree

The Rapidly-exploring Random Tree (RRT) algorithm was developed to address the limitations of the PRMs. The RRT algorithm can regenerate motion path search

[1] A paper has been published that modifies the PRM for dynamic environments, although the approach is limited to cases where the environment does not change significantly [25]

trees on the fly; the algorithm takes vehicle dynamics into consideration and has also been proven to be probabilistically complete provided certain conditions are met [33]. Where the PRM attempts to explore as much of the free space as possible in the pre-computed roadmap, the RRT algorithm attempts to capture the connectivity of the free space with as little exploration as possible. The RRT algorithm constructs a tree of feasible trajectories where the root of the tree is the initial configuration. Random configurations chosen uniformly over the configuration space are used to expand the tree to uniformly cover the free space of the problem. When a branch of the tree lands close enough to the end goal, a feasible solution is found and the program terminates.

Table 2.1: Components of RRT Algorithm

Name	**Description**	
1. State Space	A bounded manifold	$\mathcal{X} \subset \mathbb{R}^n$
2. Boundary Values	Initial and goal locations	$x_{init} \in \mathcal{X}$ and $x_{goal} \in \mathcal{X}$
3. Collision Detector	Function that determines whether constraints are satisfied	$D(x) \rightarrow \{true, false\}$
4. Inputs	Set of control actions	$\mathcal{U}$
5. Incremental Simulator	Given the current state and inputs determines future state	$S(x(t), u(t : t+\Delta t)) \rightarrow x(t + \Delta t)$
6. Metric	Specifies distance between states	$\rho : \mathcal{X} \times \mathcal{X} \rightarrow [0, \infty)$

The RRT algorithm is formulated using 6 components which are summarized in Table 2.1. First, a state space, $\mathcal{X}$, represents the space in which the problem is formulated, and could just as easily be the configuration space defined earlier. Second, a set of boundary values specify the root of the tree (x_{init}) and the center of an endgame region used to define the termination of the algorithm (x_{goal}). Within the problem formulation, the collision free subset of the state space is denoted by $\mathcal{X}_{free}$, while the subset of the state space that is defined by the location of the obstacles is denoted $\mathcal{X}_{obs}$. Third, a collision detector, which is a function that can check any given state against $\mathcal{X}_{obs}$ and return whether or not a state violates a constraint, is used to determine the feasibility of newly constructed branches of the tree. Fourth, a set of inputs, $\mathcal{U}$, represents a complete set of control actions that can effect the state. The set of inputs is bounded to reflect the capability of the system providing the control actions. Fifth, An incremental simulator uses the current state and a set of control inputs to determine a new state at a future time. Lastly, the metric, ρ, is a real valued function that establishes a measure of distance between two states [33].

To begin, a random state is chosen and the first branch of the tree is built from the root by selecting a set of control inputs that will move the vehicle from the initial state closer to the random state. The incremental simulator computes a new state from the inputs and the initial state. The collision detector checks to ensure that the path from the initial state to the final state is collision free. If the path is collision free, the new state becomes a new node on the tree. Then the new node is checked to see if it lies

within the endgame region. If the new node lies in the endgame region, the algorithm terminates with a success. On the other hand, if it does not lie in the endgame region, then a new random state is chosen and the metric is used to determine which node in the tree is closest to the new random state. The tree is once again extended from the node chosen by the metric toward the new random goal and the cycle repeats itself until a node lands in the endgame region or the algorithm exceeds a predetermined maximum number of iterations. The RRT algorithm is described using pseudocode in Table 2.2.

Table 2.2: RRT Algorithm

```
BUILD_ RRT(x_init)
1   T.init(x_init) ;
2   for k = 1 to K do
3     x_rand ← RANDOM_ STATE();
4     EXTEND(T, x_rand);
5   Return T
```

```
EXTEND
1   x_near ← NEAREST_ NEIGHBOR(x, (T));
2   if NEW_ STATE(x, x_near, x_new, u_new) then
3     T.add_ vertex(x_new);
4     T.add_ edge(x_near, x_new, u_new);
5     if x_new = x then
6       Return Reached;
7     else
8       Return Advanced
9   Return Trapped;
```

In [33] two methods of using a randomly generate samples to grow the tree are discussed. The first is called the EXTEND function which primarily uses the random sample to determine the direction to extend the tree one small incremental step, placing a node at the end of the step. If the random sample is attained before the incremental step size is reached, then the EXTEND function terminates within the incremental step size and a new node is added at the location of the random sample. If an obstacle is encountered before the incremental step size is achieved, then the EXTEND function terminates in a failure, and does not add a new node to the tree. The second function, called the CONNECT function, calls the EXTEND function in a loop that results in either the construction of a branch of the tree containing a number of consecutive nodes the end of which is the random sample, or else a branch of consecutive nodes that stops at an obstacle encountered on the way toward the random sample. In either case, every new node added to the tree lies at most an incremental step's distance away from the original tree.

Extensions and Improvements to the RRT

Methods to improve the performance RRT have been published for a number of different problems. Convergence of the algorithm can be greatly enhanced by biasing the search towards the goal [33]. Instead of exclusively using random states to expand the tree, the goal configuration or samples from a region near the goal can be chosen to grow the tree in the direction of the goal. Although a balance must be established between the use of biased samples and uniformly chosen random samples because using biased samples too often causes the tree to become entrapped in local minima [26, 50]. A second way to enhance the convergence of the algorithm involves constructing two trees, one rooted at the initial state and one rooted at the goal state in a bidirectional search. Note, for a bidirectional search, the incremental simulator used to expand the tree rooted at the goal state must propagate the states in reverse time. Using the bidirectional approach shifts the problem from connecting the tree to the endgame region to connecting the two trees together. Additional information on the bidirectional method is presented in [33]. Another way of cutting down on computation time is to employ an efficient nearest, or approximately nearest, neighbor search. A naive approach must use the metric to check the distance from each node to a new randomly generated state and then choose the minimum; more efficient techniques can be found in [23] and [1].

Additional methods have been sought out to improve the convergence the algorithm when the vehicle must transit a narrow path. One approach is to increase the number of randomly sampled states in or near passageways. These methods can be found in [44] [21] and [6]. However, there are risks associated with implementing these sampling strategies because the performance of the RRT is founded upon the fact that it rapidly and uniformly explores the free space. It is understood that changing the sampling strategy will certainly change the distribution of the free space that is searched, and may not improve the overall performance of the algorithm. Furthermore, probabilistic completeness has not been shown for the case of a non-uniform sampling strategy.

In his Master's Thesis [18], Ian Garcia attempts to improve the convergence of the RRT algorithm by testing two new connection strategies. The connection strategies work within the tradeoff between choosing fewer, more effective connections at the cost of computation time versus choosing many, simple connections which will likely take more iterations to arrive at a solution. The thesis also explores methods of smoothing feasible, open-loop solutions.

While the RRT algorithm has successfully produced results for many problems with static environments, the RRT algorithm is not designed to be directly applied to dynamic environments. Motion planning problems in dynamic environments require that the problem formulation be extended to generating a feasible path in configuration-$\times$-time space. Moving obstacles may require that the vehicle double back on its original path in order to avoid a collision. Because the RRT algorithm constructs a tree within the configuration space by expanding the node nearest to a randomly chosen sample, it is not capable of generating a path that will double back on itself, and so the algorithm is not probabilistically complete within a dynamic

environment.

2.3.3 Randomized Kinodynamic Motion Planner

In order to achieve the performance of an RRT algorithm in dynamic environments, Hsu *et al.* developed the randomized kinodynamic motion planning algorithm (RKMP) presented in [22]. The RKMP is similar to the RRT in that it also constructs a tree rooted at the initial state that terminates when it reaches an endgame region. The algorithm explores the configuration-×-time space by expanding the tree from randomly selected nodes. The tree expansion is achieved by applying a random control input for a short period of time. In this way, the tree is pushed into the free space by the use of control, instead of pulled into the free space by the selection of random target configurations. The benefit of using random configurations to pull the extension of the tree is that it promotes a uniform coverage of the configuration space. Although the control inputs used to expand the tree in the RKMP algorithm are chosen from a uniform distribution, there is no guarantee that the resulting output will uniformly cover the configuration-×-time space. In order to promote uniform coverage of the configuration-×-time space, the RKMP chooses nodes from a random distribution that is inversely related to the number of nodes present in a given region. In this way, nodes in regions that have been less explored are more likely to be chosen for expansion. Because it would be difficult to define a region around a point in configuration-×-time, the configuration-×-time space is divided into bins. In this way, the algorithm randomly selects a bin that has at least one node and then randomly chooses a node within the bin to expand. This sampling strategy approximates an ideal sampling strategy that would guarantee uniform coverage. The RKMP is outlined in pseudocode in table 2.3 [22].

Table 2.3: RKMP Algorithm

ALGORITHM	
1	Initialize $\mathcal{T}$ with m_{init}; $r \leftarrow 1$
2	**repeat**
3	Pick milestone, m from $\mathcal{T}$ with probability $\pi_T(m)$
4	$m' \leftarrow$ PROPAGATE(m, u)
5	**if** $m' \neq nil$ **then**
7	Add m' to $\mathcal{T}$; $r \leftarrow r + 1$
8	Create an arc e from m to m'; store u with e
9	**if** $m' \in$ ENDGAME **then exit** with SUCCESS
10	**if** $r = N$ **then exit** with FAILURE

2.3.4 Frazzoli's Algorithm

Inspired by Hsu's algorithm and the RRT, Frazzoli *et al.* produced another randomized trajectory planner that can handle dynamic environments [17]. Frazzoli's algorithm is able to combine the ability to plan in dynamic environments with a method that uses random configurations to pull the expansion of the tree into unexplored regions of the configuration-×-time space. Frazzoli's algorithm closely resembles the RRT with a few important differences. First, Frazzoli's algorithm assumes the existence of an obstacle free, inner-loop control strategy with an associated Lyaponov function or cost-to-go function. In a manner similar to the RRT, the obstacle free control strategy is used to attempt to connect nodes within the tree to randomly chosen configurations or the goal, while the cost-to-go function is used as the metric to determine the cost (distance or perhaps control effort) between configurations. Secondly, the expansion of the tree does not occur in incremental steps, but each branch represents a direct connection from a node to a randomly sampled configuration. It is important to recognize that using this methodology, each node of the tree represents an equilibrium or rest configuration. Furthermore, if a connection from the tree to the sample cannot be made, then the randomly chosen configuration is dropped and a new one chosen. Third, instead of attempting to expand the tree by choosing only one node to expand (the node that is nearest to a randomly sampled configuration), an attempt is made to connect the sample to the tree from every node in sequence until a successful connection is made or an attempt has been made from every node. A sequence that has been reported to produce good performance is one of increasing distance from the sample, where distance is defined by the cost-to-go metric [17]. Frazzoli's algorithm circumvents the challenging problem of defining a configuration-×-time metric by attempting to expand from every node. The attempt to expand from every node allows the algorithm to generate a path that can double back on itself, which is necessary for guaranteeing probabilistic completeness for the dynamic environment case. Frazzoli's algorithm is presented in Table 2.4.

Because the obstacles are moving, a configuration that is collision free at one point in time, may not be collision free in the following moment. To avoid generating nodes that have a very small reachable set due to a nearly immanent collision with an obstacle, each node is checked for its safety over a buffer time before being added to the tree. In theory, the buffer time could be used to compute a new solution if the vehicle must move before a final solution is achieved [17].

One drawback to generating a feasible path by stringing together a series of rest to rest maneuvers is that the final solution appears jerky and may require a significant amount of energy to execute the accelerations for all of the starting and stopping along the final path. If the obstacle free control strategy will control the system from any state to any rest configuration, then the algorithm can be improved by including secondary milestones at a random time between each node. The final solution may be improved by the addition of the milestones between nodes because it allows for sections where the trajectory moves from milestone to milestone before coming to another rest configuration [15].

Table 2.4: Frazzoli's Algorithm

```
ALGORITHM
1   Initialize Tree with initial conditions at time t_0 + θ
2   loop
3     if UPDATE-COST-ESTIMATES(Tree, root, target) = success then
4     Terminate with success
5   repeat
6     newTrajectory = EXPAND-TREE(Tree)
7     if newTrajectory ≠ failure and newTrajectory is τ safe then
8       Split newTrajectory and generate primary and secondary milestones
9       for all new milestones do
10      UPDATE-COST-ESTIMATES(Tree, node, target)
11      Prune Tree
12    until Time is up
13    if Tree.root.Upperbound < ∞ {Feasible solution found} then
14      find the child of root which gives the least upper bound on the cost-to-go
15      Tree.root ← best child
16    else if Root has children then
17      Choose a child according to a random distribution, weighted with the
          number of children in each subtree
18    else
19      propagate root for a time θ in the future
```

```
UPDATE-COST-ESTIMATES(Tree, node, target)
1   node.LowerBound ← J* (node, x, target, x).
2   Generate the obstacle-free optimal trajectory to x_f, using the control policy
    π(·, target, x).
3   Check the generated trajectory for satisfaction of the obstacle avoidance
     constraints
4     if the trajectory is collision-free then
5     node.UpperBound ← node.LowerBound.
6     while node ≠ Tree.root and node.Parent.UpperBound >
        node.Upperbound + node.incomingEdge.cost do
7       node.Parent.UpperBound ← node.UpperBound + node.incomingEdge.cost
8       node ← node.Parent
9   return succes
10  else
11    node.UpperBound ← +∞
12    return failure
```

```
EXPAND-TREE(Treee)
1   Generate a random configuration x_r
2   sort the nodes in the tree
3   for all nodes in the tree do
4     Generate a trajectory to x_r, using control policy π(·, x_r)
5     Check the generated trajectory for obstacle avoidance
6     if the trajectory is collision-free then
7       return generated trajectory
8   return failure
```

2.4 Configuration Space Considerations

One major issue concerning the successful use of a randomized trajectory planning algorithm that has not been discussed in the literature referenced is the importance of how the configuration space is defined. If the configuration space is defined as the vehicles position within a reference frame, then there may be cases where rest to rest maneuvers cannot produce a feasible path though one may exist. For example, consider a moving passage problem where a narrow passageway moves or changes shape through the course of the plan (see Figure 2-1). Let the configuration space

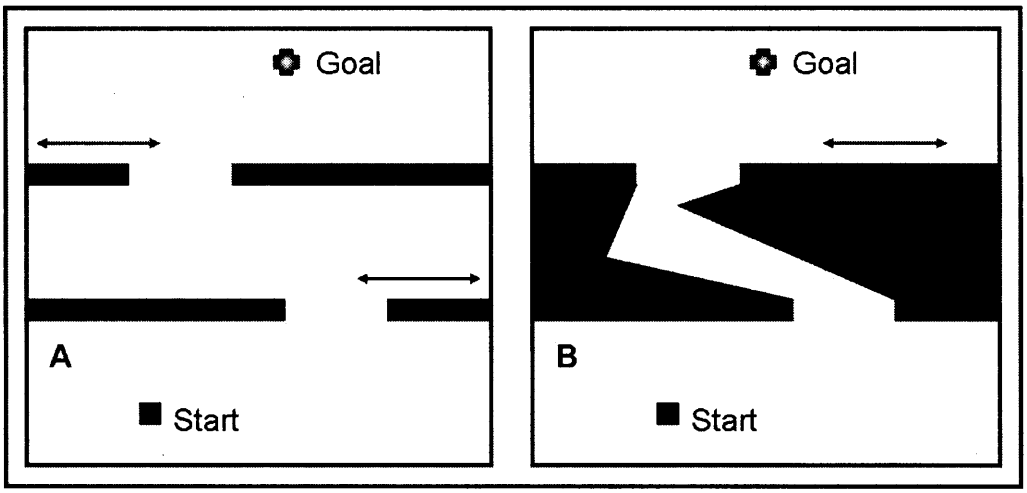

Figure 2-1: Box A depicts the sliding door problem while Box B illustrates a more tightly constrained moving passage.

be defined by all the possible positions the vehicle can with respect to a standard reference frame. If the moving passage can be negotiated by a simple rest to rest maneuver then we can expect the algorithm to find a feasible solution. On the other hand, if a simple rest to rest maneuver cannot negotiate the obstacle, then we can expect the algorithm to fail, even though the system may be capable of negotiating the obstacles with a more complicated maneuver. In [17] an application example that fits the category of a moving passage problem is analyzed using several different algorithms. The application example required a helicopter to fly through two sliding doors. Frazzoli's algorithm successfully navigated through the sliding doors primarily because a simple rest to rest maneuver could span the opening of the door as it slid back and forth. If the problem is recast as a moving passageway problem, where because of the shape of the moving passageway or the way it moved, the passage could not be negotiated by a simple rest to rest maneuver (see Figure 2-1), then Frazzoli's algorithm would fail to produce a solution. In some circumstances, the only way to negotiate a moving passage problem requires constant movement. Suppose now that the problem is recast using a different definition of configuration space. Let the configuration space now be defined by trim trajectories, where the vehicle maintains a constant velocity. In this case, maneuvers from one trim condition to the next

would be much more likely to successfully negotiate the moving passageway problem. Changing the definition of the configuration space will naturally result in a change in the data that must be stored at each node. Where the configuration space is defined by the velocities of the vehicle, each node would have to store the configuration, the position as it entered the configuration and if applicable, the length of time spent in that configuration.

2.5 Extensions of Randomized Trajectory Planning in Dynamic Environments

Though both Frazzoli's and Hsu's algorithms solve the problem of determining a feasible, open-loop trajectory from an initial state to a goal configuration in the presence of moving obstacles, there are a number of extensions to the moving obstacle problem that remain to be explored. A straight forward extension to the problem is to have a goal configuration that also moves, which is also referred to as an intercept problem. If the trajectory of the goal is perfectly known, then Hsu's algorithm may be directly applied where each new node is tested to see if it matches with the goal's configuration-$\times$-time. It is believed that Frazolli's algorithm may also be directly applied with one minor change. Where in the original algorithm there is only one obstacle free controller, a second obstacle free controller that is capable of intercepting a moving goal should be added. A switching logic may then be employed to choose which obstacle free guidance law used depending upon whether a connection is sought with a random configuration or the goal. For an illustration of a moving goal problem see Figure 2-2.

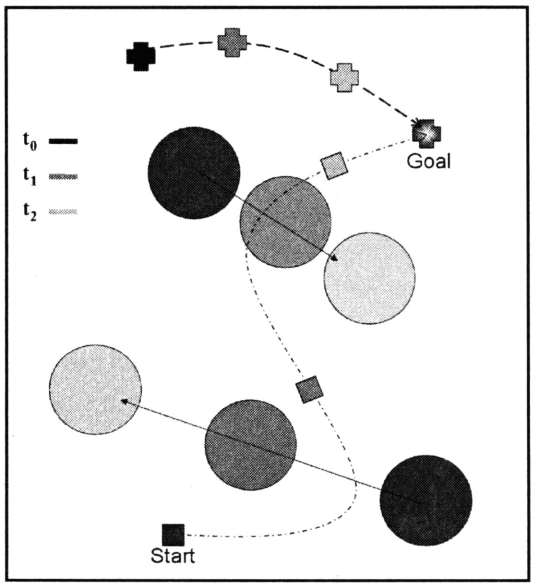

Figure 2-2: An intercept problem with moving obstacles.

A second extension to the moving obstacle problem is to require a vehicle maintain a relative position or orientation with respect to a moving object. This extension will be referred to as the tracking problem. Consider the problem where two robots must continually point antennas toward one another as they navigate an obstacle field. Provided that the location of each robot relative to the other is well known, then both Frazzoli's and Hsu's algorithms can be directly applied, by folding the tracking portion of the problem in as a constraint. One may notice that the constraints specified in a tracking problem often resemble the moving passage problem discussed earlier in the context of choosing the description of the configuration space.

An extension to the moving obstacle problem that is not directly solved by Frazzoli's or Hsu's algorithms is the generation of a motion plan in the presence of moving obstacles that is robust to errors and uncertainties. Both Frazzoli's and Hsu's algorithms are considered open-loop planners because they formulate a single plan with the expectation that the system and the environment are modeled accurately enough that the execution of the plan will be successful. The open-loop planning concept is very similar to using an open-loop controller to drive a system to a desired state. An open-loop controller calculates the inputs required to drive the system to a desired state and feeds the inputs to the system without regard to how closely the system output matches the expected output. Usually the system outputs do not exactly align with the expected outputs because of various errors and uncertainties. In the same way, the performance of a system executing an open-loop plan will be subject to errors in the model used to generate the plan and uncertainties in the environment.

As an example, consider the intercept problem mentioned earlier, and assume there are no modeling errors or uncertainties in the environment except for some uncertainty in the goal's initial position. Assume also that the dynamics by which the goal moves are perfectly known and are a function of the goal's position. The goal location at a given time is found by integrating the dynamics of the goal from the initially uncertain position. Because the dynamics of the goal are a function of its position, the uncertainty in the goal's location will grow in time. At the beginning of the engagement the uncertainty in the goal's location may be fairly low and the likelihood of an open-loop plan successfully intercepting the goal could be fairly high. However, as time goes on, the uncertainty in the goal's location will grow and the likelihood of successfully intercepting the goal by using an open-loop plan will diminish. For an illustration refer to Figure 2-3.

2.6 Robust Motion Planning

In the realm of system control, the method of feeding the output errors of the system back to the controller is known as a feedback control strategy. Feedback controllers tend to be much more robust to errors and uncertainties than their open-loop counterparts. In the same way, if a feedback approach can be taken with motion planning in the presence of errors and uncertainties, then the resulting motion plans can be expected to be more robust. In regards to motion planning, the problematic result of the error sources is that the trajectory taken by the system does not precisely align with

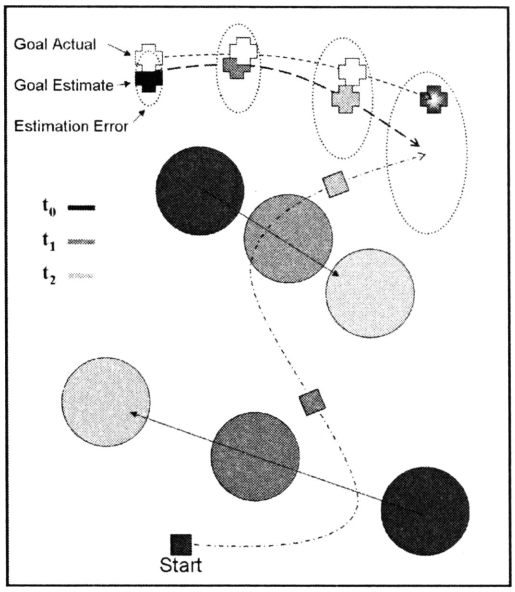

Figure 2-3: An intercept problem with error-prone estimation.

the planned trajectory. In order to account for errors within the plan, individual error sources must be identified and an upper bound of their contribution to the system's deviation from the plan must be estimated. Combining the error sources maximum contribution to the deviation form the planned trajectory provides an estimate for the maximum deviation from the path. Knowing the maximum deviation allows the planner to artificially enlarge the constraint boundaries so that the resulting path will clear each obstacle by at least the distance of the maximum deviation.

Before formulating a robust motion planning strategy, it is important to identify the sources of error and uncertainty and how they affect the performance of the system. First, consider the error generated by mismodeling the system. A randomized trajectory planner must contain a model of the system to construct the tree that searches the state space. When an open-loop plan is formed, the planner supplies a controller with the open-loop control inputs that are expected to carry the system through the plan. Because mismodeling errors and unexpected environmental forces are generally anticipated, a feedback controller can be implemented to force the system to closely follow the trajectory provided by the planner. Even with the use of a feedback controller it is expected that the system will not track the trajectory exactly. The degree to which the controller allows the system to drift from the trajectory provided by the planner will be referred to as the controller error. In summary, the source of controller error is mismodeled dynamics, sensor noise, and unexpected environmental forces while the effect of the error is a deviation from the planned trajectory.

Next, consider the fact that a navigation system provides estimates of the system state and information about the environment based on measurements from sensors that have a limited accuracy. Therefore, the data provided by the navigation system

is also error prone. Navigation errors effect the performance of both the controller and the planner. The system state estimates are fed back to the controller as a measure of the system output. If the controller is not subject to errors, then the navigation system's state estimates would match the plan exactly. However, because the navigation system's state estimates tend be different then the system's true states, the system is expected to deviate from the planned trajectory. Navigation data is also used by the planner to generate a description of the system's initial state and to define constraint boundaries. As a result of navigation errors, the true constraint boundaries may not be perfectly represented in the planner, so that a feasible plan formed that moves the system near a predicted constraint boundary may in reality cause the system to run into the true constraint boundary [43].

In order to guarantee that the system's deviation from the trajectory will not result in the violation of a constraint, the planner must construct a plan according to constraint boundaries that have been artificially enlarged by the maximum predicted deviation. A maximum predicted deviation is derived by adding together the maximum effect of each individual error. Some errors and their maximum effect on the system's deviation from the planned path can be expected to remain nearly constant in time. Making some reasonable assumptions about the feedback controller, it can be expected that the controller will be able to keep the system close to the planned trajectory throughout the execution of the plan. Similarly, if the navigation system has regular access to accurate updates, the navigation error can be expected to remain bounded by a constant. Other errors, like the propagation errors, are expected to grow in time. Naturally the longer the plan generated the larger the effect of propagation errors. The consequence of creating a plan robust to propagation errors is that at some point the boundaries of the constraints will grow to consume the entire free space, making a successful plan impossible. With a robust implementation there will be a trade-off between amount of free space surrendered to artificial constraint boundaries and the length of time for which the plan can be generated. In cases where the goal and the environment are well known, propagation errors may have a limited role if any and the free space trade-off problem may not arise. However, in many intercept or tracking problems, the goal and/or the environment are often subject to propagation errors and the free space trade-off problem must be addressed.

Most randomized trajectory planning algorithms admit slow convergence to a solution when the problem is high constrained or involves narrow passages. Therefore, the degree to which the free space can be infringed upon by artificial constraints while not significantly affecting the convergence of the planner must be decided for each problem encountered. Expansive problems may permit a great deal of constraint enlargement before the planner's convergence rate is significantly impacted, while the opposite is true of problems that are already tightly constrained (see Figure 2-4).

Specific methods for determining how much artificial constraint boundary expansion to allow, along with other specific details will be provided in Chapter 4. For now, the general framework will be described. First, a pre-planning function using prior error bound analysis determines a safe planning horizon and reasonable constraint boundaries. The safe horizon and the boundary definitions are sent to the planner which generates a plan. The plan is then provided to an inner-loop feedback

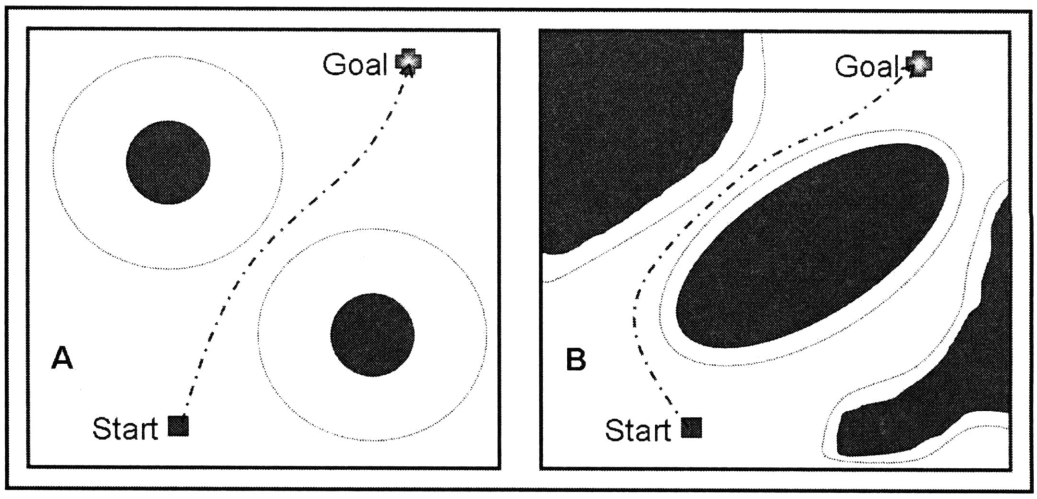

Figure 2-4: Box A depicts an expansive problem where the constraints can be artificially enlarged a great deal before severely hampering the convergence rate of the algorithm. Box B illustrates a more tightly constrained problem where the constraints cannot be enlarged much before the problem becomes infeasible.

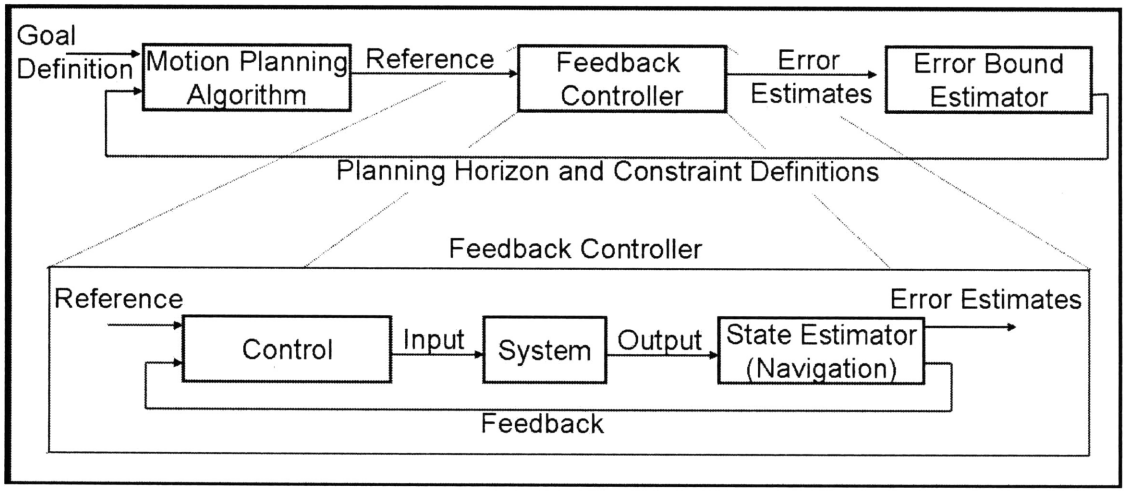

Figure 2-5: The framework of a robust motion planning strategy

controller which executes the plan. A navigation system provides data to a feedback controller while also supplying the preplanner with current estimates on the system and environment. As time approaches the first safe horizon time provided to the planner, the pre-planning function provides an updated planning horizon time with constraint boundary definitions. The robust planning architecture contains an outer planning loop and an inner execution loop (see Figure 2-5). Chapter 3 describes the inner-loop mechanisms while Chapter 4 goes into detail on the implementation of the outer planning loop.

Chapter 3

Inner-loop Description

The OTE attitude guidance problem presents a challenging test case for the proposed robust randomized trajectory planning approach introduced in Chapter 2. The post-acquisition phase of the OTE mission naturally falls into the category of a tracking problem where the constraints placed on the system form narrow moving passageways. Furthermore, the ultimate success of the tracking experiment hinges on the system's ability to remain within the constraints at all times possible.

This chapter presents the description of the inner-loop experimental setup used to test and refine the robust planning methodology introduced in Chapter 1. The description of the inner-loop setup used in this experiment will begin with a report of the satellite system hardware and environment specifications. Hardware and environment particulars dictate interfacing requirements and affect the performance bounds of the navigation and the control systems. Following the system and environment description, will be an explanation of the navigation system and its associated error sources. Discussion and some basic results will also be presented on the methods used to obtain rotational state estimates and translational state predictions. Finally, a third section that mirrors the structure of the navigation section will present the obstacle-free tracking controller used to implement the planned trajectory. Error contributions associated with the tracking controller will also be discussed.

3.1 System and Environment

The two satellites in the OTE each contain the following sensor suite and hardware components: a primary camera for providing line-of-sight measurements of the tracked object, a star camera to provide three-axis attitude updates, a communications antenna which permits the transfer of data between satellites, a GPS receiver to provide own-ship translational position and velocity updates, an IMU that supplies angular velocity data, and reaction wheels which provide the torque necessary to steer the attitude of the satellite. Table 3.1 reports the location, specifications and the associated error quantities of each sensor and hardware item, while Figure 3-1 provides an illustration of the sensors and hardware.

Because each component is rigidly mounted to the frame of the satellite, rigid

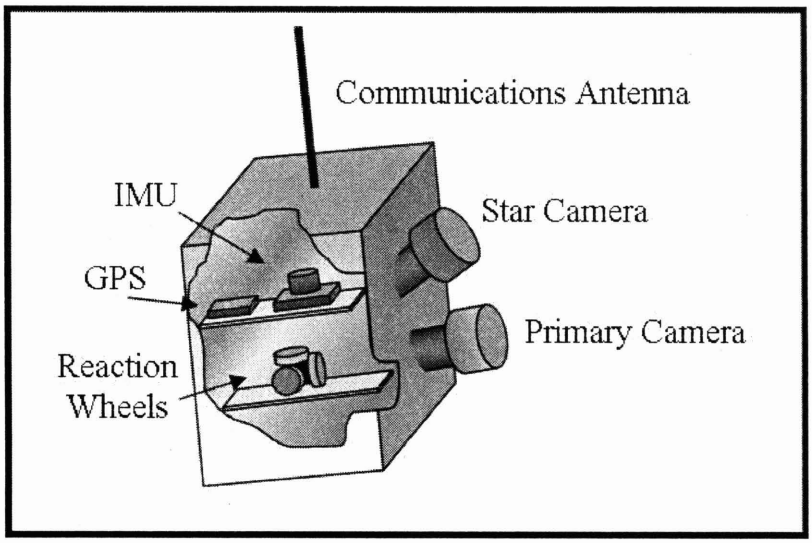

Figure 3-1: The sensors and hardware of each OTE satellite.

body dynamics can be used to represent the motion of the satellite. The rigid body equations of rotational motion are given in Equations 3.1 and 3.2, where $\mathbf{J}$ is the moment of inertia tensor, $M(\mathbf{u}^B)$ represents the moments applied to the satellite from control actions, and $M(\mathbf{v}^B)$ represents moments resulting from environmental disturbances. Note also that $\omega^B_{B\_I}$ is the angular velocity vector of the satellite, given in the body frame, and q^I_B is the attitude quaternion of the satellite.

$$\mathbf{J}\dot{\omega}^B_{B\_I} = -\omega^B_{B\_I} \times \mathbf{J}\omega^B_{B\_I} + M(\mathbf{u}^B, \mathbf{v}^B) \tag{3.1}$$

$$\dot{q}^I_B = \frac{1}{2} q^I_B \otimes \omega^B_{B\_I} \tag{3.2}$$

The moment of inertia tensor used in the experiment is given in Equation 3.3.

$$\mathbf{J} = \begin{bmatrix} 120 & 11 & 12 \\ 10 & 125 & 9 \\ 13 & 8 & 115 \end{bmatrix} \tag{3.3}$$

Recall that the operating limits of the primary camera, star camera, and communications antenna define pointing constraints which are formulated as inclusion or exclusion cones. The pointing constraints are listed in Table 3.3 according to each sensor. See Figure 3-2 for an illustration of the inclusion and exclusion cones.

The robust randomized trajectory planning methodology is tested using three engagement scenarios. For each scenario, the satellites are phased in the same orbit and the orbit of the object to be tracked is different orbit. Table 3.4 lists the scenarios along with the associated orbital elements and the engagement times.

Table 3.1: Hardware Specifications Used in OTE

Item	**Location (B)**	**Specifications**	**Symbol**	**Errors** 1σ
Primary Camera	Bore-sight $[1\ 0\ 0]'$	10 degree FOV	NA	NA
Star Camera	Bore-sight $[0.7660\ 0\ 0.6428]'$	22 degree FOV 7 deg/s max rate	$\sigma_{sc\_p,y}$ σ_{sc_r}	1 arcsec Pitch and Yaw 5 arcsec Roll
Com. Antenna	Antenna Axis $[0\ 0\ 1]'$	60 deg cone of insensitivity	NA	NA
GPS	Internal	NA	σ_{Sp} σ_{Sv}	16 m Position Error 0.4 m/s Velocity Error
IMU	Internal	NA	σ_b σ_{sf} σ_{ma} σ_{rw}	1 deg/hr Bias 100 ppm Scale factor 100 μ-radians Misalignment 0.07 deg/$\sqrt{\text{hr}}$ Angle Random Walk
Reaction Wheels	Internal	Max rate: 3 deg/s Max acceleration: 0.15 deg/s^2	NA	Not modeled

Table 3.2: Environmental Error Source

Source	**Symbol**	**Error** 1σ
Object Translational State	σ_{Op} σ_{Ov}	1e5 m Position Error 1e3 m/s Velocity Error

Table 3.3: Inclusion and Exclusion Constraint Definitions

Hardware Item	**Inclusion or Exclusion**	**Defining Vector**	**Half-angle**
Primary Camera bore-sight	Inclusion	Vector from satellite to object	5 degrees
Star Camera bore-sight	Exclusion	Vector from satellite to Sun Vector from satellite to Earth Vector from satellite to Moon	30 degrees 75 degrees 10 degrees
Positive and Negative Communications Antenna axis	Exclusion	Vector from satellite to 2$^{\text{nd}}$ satellite	30 degrees

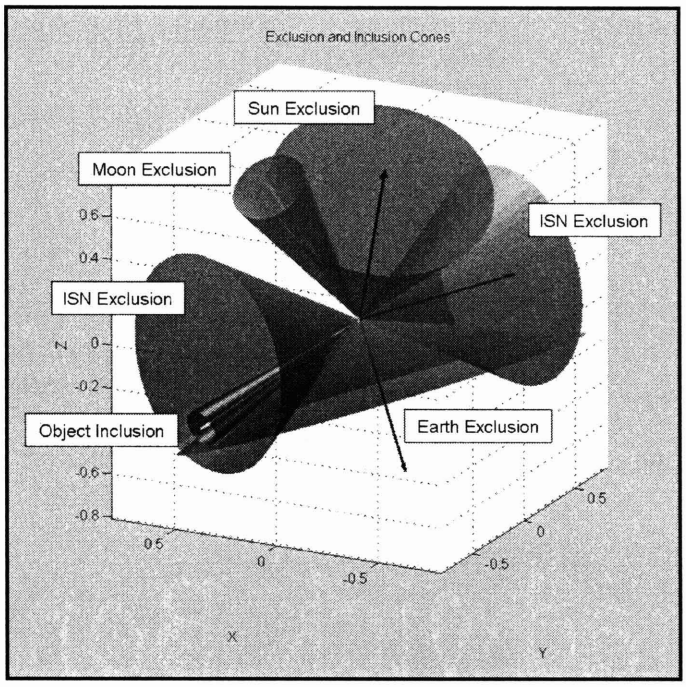

Figure 3-2: The inclusion and exclusion cones of the attitude guidance problem.

Table 3.4: Engagement Scenario Orbits

Scenario		Semi-Major Axis(km)	Eccentricity	Inclination (deg)	RAAN (deg)	Arg. of Perigee (deg)	True Anomoly
1	Satellite 1	7004	0.0701	180	0	276.921	38.131
	Satellite 2	7004	0.0701	180	0	276.921	17.968
	Object	7200	0.005	70	45	20	331.32
2	Satellite 1	7004	0.0701	180	0	276.921	27.781
	Satellite 2	7004	0.0701	180	0	276.921	6.94
	Object	7200	0.005	150	45	20	320.652
3	Satellite 1	7004	0.0701	180	0	276.921	351.36
	Satellite 2	7004	0.0701	180	0	276.921	325.68
	Object	7700	0.005	100	90	20	325.921

3.2 Navigation System

A navigation system provides information about the state of the system and/or information on the state of the environment. For this experiment, the information provided on the system includes estimates of the satellite's rotational and predicted translational states along with the associated estimation error covariance measurements. The environment data includes the ephemerides of the Sun, Earth, and Moon, and a prediction of the object's translational state with an associated estimation error covariance. The translational state and estimation error covariance of the second satellite are relayed from the navigation system of the second satellite through the communications link for use by the preplanning function.

The navigation system designed for this experiment reports system and environment information to both the outer-loop, preplanning function, and the inner-loop, control system (refer back to Figure 2-5). The estimated rotational state of the satellite is used to provide feedback to the inner-loop control system. The estimation error covariance of the satellite's rotational state is not used in this simulation, though it could be used to measure the confidence in the rotational state estimate. The predicted translational states of both of the satellites and the object are used by the outer-loop preplanning function along with the estimation error covariance measurements associated with the predicted translational state of the satellite and the object. The following is a summary of how the rotational state estimates and predicted translational state information is worked out in simulation.

3.2.1 Rotational State Estimates

In order to simulate realistic rotational state estimates from sensor data, a discrete-time Kalman filter is utilized to combine simulated IMU and star camera data. The Kalman filter is an optimal linear filter that can be used to form optimal filter state estimates by combining measurements from multiple sensors [19]. The use of a Kalman filter is an approach that would likely be taken to generate navigation information onboard a real satellite. This subsection begins by introducing Kalman filtering preliminaries and two different implementations of the Kalman filter, then general Kalman filter equations are provided, and finally the section concludes with the specific equations used to generate the rotational state estimate for this experiment and a report of the navigation system results.

General Kalman Filter Preliminaries: The discrete-time Kalman filter can be described in terms of three estimation related processes: initialization, propagation, and measurement update/reset. To begin, the Kalman filter initializes the filter state and covariance estimates by populating them using initial estimated values. The Kalman filter uses a plant/process model to propagate the filter state via a series of discrete time propagation steps to a measurement time. Measurements from sensors at measurement times correct the predicted state estimates, where a set of calculations are used to update the filter state and covariance estimates. At the measurement time, there may also be the option to reset certain variables depending upon how the

Kalman filter is implemented. Following the measurement/resent process, the filter propagates the updated filter state estimate to the next measurement time and the cycle repeats [19].

When using Kalman filter equations there are at least two cases where variables must be distinguished by careful notation. First, it is important to recognize that the filter state is different then the system state. In the context of this experiment, the system state is the rotational state of the satellite, namely the attitude and angular rates. The filter state is often a vector of estimated error values, as is the case in the Kalman filter used in this experiment. In this section the system state will be distinguished from the filter state by adding a subscript, s, to the front of the variable (i.e. ${}_{s}\mathbf{x}$). Secondly, the filter state and covariance estimates before the measurement/reset process must be distinguished from the filter state and covariance estimates after the measurement/reset process. A filter state estimate at time t_k, that is the result of j measurements is given as $\mathbf{x}_{k|mj}$. Therefore, during the measurement/reset process $\mathbf{x}_{k|mj}$ becomes $\mathbf{x}_{k|mj+1}$, because the entire measurement process occurs within time-step t_k.

Two Implementations: There are two ways of implementing a Kalman filter that will be described here as the feedforward method and the feedback method. The methods are differentiated by how the system output is corrected by the filter output. The implementation method is called feedforward if the Kalman filter uses uncorrected system data to generate the filter estimate and then corrects the system output before the information is sent elsewhere. The implementation method is called feedback if the system data is corrected by the previous filter state estimate before being used by the filter to generate the next filter state. For an illustration of the feedforward and feedback methods refer to Figure 3-3.

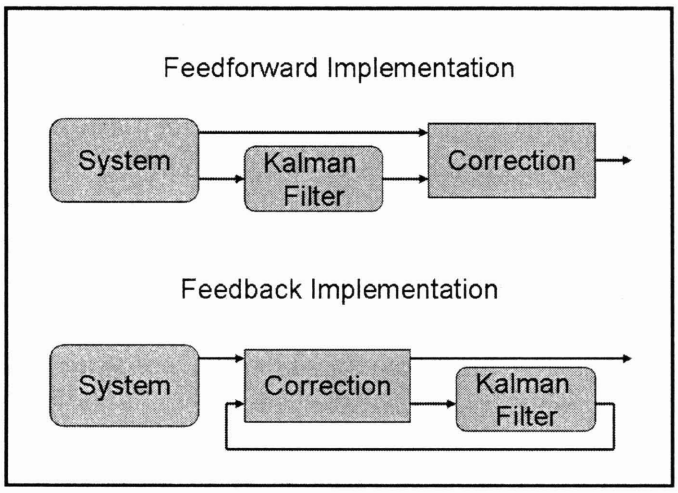

Figure 3-3: The feedforward and feedback methods of Kalman filtering

In the feedforward method, the error estimate accumulates in the filter state. Consider the case where a feedforward Kalman filter is used to estimate and correct

for errors that grow over time. Ideally, the values of the filter state grow to match the errors generated within the system. However, at some point, the size of values in the filter state could invalidate the linear assumptions made in the filter (recall that the Kalman filter is the optimal linear filter). On the other hand, in the feedback method, the value of the error estimate does not accumulate in the filter state but rather in a set of correction variables which are used to correct the system output data before it is processed by the filter. In this way, the filter state estimate reflects the error growth since the last measurement instead of reflecting the error growth since the filter was initialized. The feedback method keeps filter error state values low, which works to justify the linear assumptions made within the filter.

Discrete Kalman Filter

In this subsection the discrete state space model is derived from a continuous state space representation. Also presented here are approximations of the state transition matrix and discrete process noise covariance matrix which do not require matrix exponentiation. The subsection concludes with the discrete time Kalman filter equations. For more information see [19]

Continuous time model: The continuous time state propagation equation is generally formulated as Equation 3.4, where the initial condition is $\mathbf{x}(t_0) = \mathbf{x}_0$, and where, $\mathbf{x}$ is the n-dimensional state vector, $\mathbf{F}(t)$ is the $(n \times n)$ time-varying dynamics matrix, $\mathbf{G}$ is an $(n \times m)$ matrix, and $\mathbf{q}_c$ is the $(m \times 1)$ continuous time process noise.

$$\dot{\mathbf{x}} = \mathbf{F}(t)\mathbf{x} + \mathbf{G}\mathbf{q}_c \tag{3.4}$$

The process noise is assumed to be a zero-mean white process for which the conditions in Equation 3.5 hold.

$$\begin{aligned} \mathrm{E}\{\mathbf{q}_c\} &= 0 \\ \mathrm{E}\{\mathbf{q}_c(t+\tau)\mathbf{q}_c(t)'\} &= \mathbf{Q}_c\delta(\tau) \end{aligned} \tag{3.5}$$

The term $\mathbf{Q}_c\delta(\tau)$ is the $(m \times m)$ continuous time noise spectral density and $\delta(t)$ is the Dirac delta function. The measurement equation is inherently discrete and given by Equation 3.6.

$$\mathbf{z_k} = \mathbf{H}\mathbf{x_k} + \mathbf{r_k} \tag{3.6}$$

In Equation 3.6 $\mathbf{H}$ is a $(l \times n)$ matrix relating the $(l \times 1)$ measurement vector, $\mathbf{z}$, to the state, $\mathbf{x}$; and where $\mathbf{r}$ is a $(l \times 1)$ measurement noise vector. The measurement noise is assumed to be a zero-mean white process for which the conditions of Equation 3.7 hold true. The term $\mathbf{R}$ is the measurement noise covariance, and $\delta_{i,j}$ is the Kronecker delta function.

$$\begin{aligned} \mathrm{E}\{\mathbf{r_k}\} &= 0 \forall k \\ \mathrm{E}\{\mathbf{r}_i\mathbf{r}_j'\} &= \mathbf{R}\delta_{i,j} = \begin{Bmatrix} \mathbf{R} & \text{if } i = j \\ 0 & \text{otherwise} \end{Bmatrix} \end{aligned} \tag{3.7}$$

Discrete time model: Because a computer implementation of a state-space model is necessarily discrete-time, the discrete time state-space model is developed from the continuous time state-space equations, as in Eqaution 3.8.

$$\mathbf{x}_{k+1} = \Phi(k+1,k)\mathbf{x}_k + \int_{t_k}^{t_{k+1}} \Phi(k+1,s)\mathbf{G}\mathbf{q}_c(s)ds \tag{3.8}$$

In Equation 3.8 the initial condition is $\mathbf{x}(k=0) = \mathbf{x}_0$, where $\Phi(k+1,k)$ is the state transition matrix and the integral is the discrete-time process noise. Note that if $\mathbf{F}$ is constant then the transition matrix is exactly the matrix exponential given in Equation 3.9

$$\Phi(k+1,k) = e^{\mathbf{F}\Delta t}, \quad \Delta t = t_{k+1} - t_k \tag{3.9}$$

However, if $\mathbf{F}$ is time varying then the transition matrix is not (exactly) the matrix exponential. It is common practice to restrict the time step Δt to be small enough so that $\mathbf{F}(t)$ can be considered constant on (t_k, t_{k+1}).

$$\mathbf{F}(t) \approx \mathbf{F}(t_k), \quad t_k < t < t_{k+1} \tag{3.10}$$

With respect to the discrete time approximation used in Equation 3.10, the matrix exponential gives a good approximation to the transition matrix, which is provided in Equation 3.11.

$$\Phi(k+1,k) \approx e^{\mathbf{F}_k\Delta t} \tag{3.11}$$

Because the matrix exponentiation required to obtain $\Phi(k+1,k)$ is computationally costly, it is desirable to approximate $e^{\mathbf{F}_k\Delta t}$ by a truncated Taylor series expansion. Assuming $\mathbf{F}(t)$ is constant on the interval (t_k, t_{k+1}) and equal to $\mathbf{F}_k$, then $\Phi(k+1,k)$ is approximated to third-order terms by Equation 3.12.

$$\Phi(k+1,k) \approx \mathbf{I} + (\mathbf{F}_k\Delta t) + \frac{1}{2}(\mathbf{F}_k\Delta t)^2 + \frac{1}{6}(\mathbf{F}_k\Delta t)^3 \tag{3.12}$$

The $(n \times 1)$ discrete time process noise $\mathbf{q}_d$ is then given by Equation 3.13

$$\mathbf{q}_d(k+1,k) = \int_{t_k}^{t_{k+1}} \Phi(k+1,s)\mathbf{G}\mathbf{q}_c(s)ds \tag{3.13}$$

Note that $\mathbf{q}_d$ is a zero-mean process with covariance $\mathbf{Q}_d$, which is shown in Equation 3.14.

$$\begin{aligned} \mathrm{E}\{\mathbf{q}_d(k+1,k)\} &= 0 \\ \mathrm{E}\{\mathbf{q}_d(j+1,j)\mathbf{q}_d(k+1,k)'\} &= \mathbf{Q}_d\delta_{j,k} \end{aligned} \tag{3.14}$$

The $(n \times n)$ discrete-time process noise covariance is given exactly by Equation 3.15.

$$\mathbf{Q}_d(k+1,k) = \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} \Phi(k+1,s_1)\mathbf{G}\mathbf{Q}_c\delta(s_2-s_1)\mathbf{G}'\Phi(k+1,s_2)'ds_1ds_2 \quad (3.15)$$

To obtain a computationally suitable approximation of $\mathbf{Q}_d$, substitute the truncated Taylor series for Φ (given in Equation 3.12), in place of Φ in Equation 3.15. Solving the integral with the approximate Φ in place yields Equation 3.16 to third-order terms.

$$\begin{aligned} \mathbf{Q}_d(k+1,k) &\approx \mathbf{M}\Delta t + \frac{1}{2}[\mathbf{M}\mathbf{F}_k' + \mathbf{F}_k\mathbf{M}](\Delta t)^2 + \ldots \\ &\quad \frac{1}{6}[\mathbf{M}(\mathbf{F}_k')^2 + 2\mathbf{F}_k\mathbf{M}\mathbf{F}_k' + (\mathbf{F}_k)^2\mathbf{M}](\Delta t)^3 \qquad (3.16) \\ \text{where } \mathbf{M} &= \mathbf{G}\mathbf{Q}_c\mathbf{G}' \end{aligned}$$

To simplify the notation for future reference, the discrete state transition matrix and the discrete-time process noise covariance will be referred to as in Equations 3.17 and 3.18.

$$\Phi_k = \Phi(k+1,k) \quad (3.17)$$

$$\mathbf{Q}_k = \mathbf{Q}_d(k+1,k) \quad (3.18)$$

With the discrete state space representation established, the following is a summary of the discrete Kalman filter equations.

Initialization process: The Kalman filter state and covariance are initialized with estimates of the initial values, as in Equation 3.19.

$$\begin{aligned} \text{Initial state estimate} \quad & \hat{\mathbf{x}}_{0|m0} = \mathbf{x}_0 \qquad (3.19) \\ \text{Initial estimation error covariance} \quad & \mathbf{P}_{0|m0} = \mathbf{P}_0 \end{aligned}$$

Propagation process: The filter state and error covariance are propagated between measurement updates according to Equations 3.20 and 3.21. The propagation process usually occurs at a high rate to minimize the error associated with discrete representation of a continuous process.

$$\Phi_k : \text{given by Equations 3.12 and 3.17.}$$

$$\hat{\mathbf{x}}_{k+1|mj} = \Phi_k\hat{\mathbf{x}}_{k|mj} \quad (3.20)$$

$$\begin{aligned} \mathbf{Q}_k &: \text{given by Equations 3.16 and 3.17} \\ \mathbf{P}_{k+1|mj} &= \Phi_k \mathbf{P}_{k|mj} \Phi_k' + \mathbf{Q}_k \end{aligned} \tag{3.21}$$

Measurement update/reset process: The filter state and estimation error covariance are updated at the measurement time via Equations 3.22 and 3.23 respectively through the use of the Kalman gain, which is provided in Equation 3.24. Note that here the term update refers to the correction of the filter state and covariance values using the measured data. The measurement process is inherently discrete and usually occurs at a much lower rate the propagation process.

$$\hat{\mathbf{x}}_{k+1|mj+1} = \hat{\mathbf{x}}_{k+1|mj} + \mathbf{K}_{k+1}[\mathbf{z}_{k+1} - \mathbf{H}_{k+1}\hat{\mathbf{x}}_{k+1|mj}] \tag{3.22}$$

$$\mathbf{P}_{k+1|mj+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})\mathbf{P}_{k+1|mj}(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})' + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}' \tag{3.23}$$

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1|mj}\mathbf{H}_{k+1}'[\mathbf{H}_{k+1}\mathbf{P}_{k+1|mj}\mathbf{H}_{k+1} + \mathbf{R}_{k+1}]^{-1} \tag{3.24}$$

If the feedforward implementation is used, then the measurement/reset process is finished with the state and estimation error covariance updates. However, if the feedback implementation is used then the system state and filter states need to be reset. Resetting the filter state involves transferring data from the filter state to a set of variables which are used to correct the output of the system. Once the transfer is complete, the filter state is reinitialized to capture the change in the filter state values that will occur between the measurement just taken and the next measurement. The reset equations for the feedback implementation are given in Equation 3.25.

$$\begin{aligned} {}_s\bar{\mathbf{x}} &\leftarrow R_{sys\_reset}(\bar{\mathbf{x}}, \hat{\mathbf{x}}_{k+1|mj+1}) \qquad (3.25) \\ \hat{\mathbf{x}}_{k+1|mj+1} &\leftarrow (\mathbf{I} - \mathbf{A}_2)\hat{\mathbf{x}}_{k+1|mj+1} \qquad (3.26) \end{aligned}$$

The system reset function, R_{sys_reset}, represents the calculations required to re-compute the system state, ${}_s\mathbf{x}$, making corrections for error estimates, $\hat{\mathbf{x}}_{k+1|mj+1}$. These calculations include unit conversions between system and filter states, account for the fact that a filter state may represent a combination of system errors, and also account for the fact that only some filter states may be used in the system reset. If R_{sys_reset} is linear it can be formulated as a matrix that is referred to as $\mathbf{A}_1$. The matrix $\mathbf{A}_2$ accounts for the fact that only some of the filter states may be used for the system reset. Note that if all of the filter states are used in the system reset, then $\mathbf{A}_2$ is the identity matrix and the filter state estimate after the reset is equal to zero.

Specific Implementation

The information and equations required to utilize a feedback Kalman filter for the specific application to determining the rotational state of a satellite are conveyed in the following section. First, a description of expected hardware output is provided followed by some basic equations used to develop the filter state values. Then the filter dynamics are presented along with the equation defining a measurement update. Initial conditions are then provided and finally the reset and correction equations are

Table 3.5: Estimator Measurements

Hardware	Output	Variable
IMU	Incremental angles specifying rotations about each of the IMU axes.	$\begin{bmatrix}\Delta\alpha_1 \\ \Delta\alpha_2 \\ \Delta\alpha_3\end{bmatrix}^B$
Star Camera	Attitude quaternion specifying the rotation from the SC frame to the I frame.	$\hat{\mathbf{q}}^I_{SC}$

described.

Derivation of Measurement and Dynamic Model: Typically, an IMU will supply incremental angles thru which the body rotates between two time ticks, specifying rotations about each IMU axis in the in IMU frame. For this experiment, it is assumed that the IMU provides incremental angles and that the IMU frame is identical to the body frame. The incremental angles are used to generate angular velocity values specifying the rotation rates of the satellite with respect to the inertial frame formulated in the body frame. The attitude of the satellite is measured by the star camera, which is assumed to supply a quaternion specifying the rotation from the star camera frame to the inertial frame. Table 3.5 lists the hardware associate with attitude determination and provides a description of the raw output data expected from each instrument.

The uncorrected hardware output is first converted to derived quantities that are more useful in equations used to correct the hardware output and serve as input to the Kalman filter. Equation 3.27 is used to convert the incremental angles provided by the IMU into an angular velocity vector, while the quaternion supplied by the star camera is converted into a direction cosine matrix (DCM)[1] and a simple rotation is performed to transform the measurement into the body frame (see Equation 3.28).

$$\omega^B_{B\_I} \approx \frac{\Delta\alpha^B}{\Delta t_{IMU}} \tag{3.27}$$

$$\hat{\mathbf{T}}^I_B = \mathbf{T}^{SC}_B \hat{\mathbf{T}}^I_{SC} \tag{3.28}$$

The prime error value estimated by the Kalman filter is the error in the derived attitude. To understand how the filter can provide estimates for the attitude error, consider the rotation of a reference frame by small angles. A 3-2-1 (y-p-r) sequence of single axis Euler rotations to transform from frame A to frame B is given in Equation 3.29. If the angles θ_1, θ_2, and θ_3 are small and small angle approximations are used, then the order of rotations does not matter and Equation 3.29 can be rewritten as

[1]For equations that convert from quaternions to DCMs and vice versa consult a reference book such as [52].

Equation 3.30.

$$\mathbf{T}_A^B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 \\ 0 & -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix} \begin{bmatrix} \cos\theta_3 & \sin\theta_3 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.29}$$

$$\mathbf{T}_A^B \approx \mathbf{I} - [\theta^B \times] \tag{3.30}$$

From Equation 3.30 the attitude error, ϕ^B, can be defined in terms of the true attitude $\mathbf{T}_B^I$ and the attitude estimate $\hat{\mathbf{T}}_B^I$. Equation 3.31 gives the attitude error in skew symmetric form.

$$[\phi^B \times] \approx \mathbf{I} - \mathbf{T}_B^{I\prime}\hat{\mathbf{T}}_B^I \tag{3.31}$$

Note that a DCM is an orthonormal matrix whose inverse is equal to its transpose. Now let $\omega_{B\_I}^B$ represent the error-free angular velocity vector of the IMU frame with respect to the inertial frame expressed in the IMU frame and let $\hat{\omega}_{B\_I}^B$ represent the error-prone IMU measurement of $\omega_{B\_I}^B$, including IMU gyro errors. Then the error $\delta\omega_{B\_I}^B$, in $\hat{\omega}_{B\_I}^B$ is defined in Equation 3.32.

$$\delta\omega_{B\_I}^B = \hat{\omega}_{B\_I}^B - \omega_{B\_I}^B \tag{3.32}$$

The IMU angular velocity measurement errors, $\delta\omega_{B\_I}^B$, can be decomposed into a linear combination of the contributing error sources. Equation 3.33 delineates the contributing error sources in equation form where $\mathbf{b}^B$, $(\mathbf{D}_\omega\mathbf{s})^B$, $([\omega_{B\_I}^B\times]\mathbf{m})^B$, and $\mathbf{w}^B$ are vectors whose components represent the bias, scale factor, misalignment, and process noise along each axis of the IMU frame. $\mathbf{D}_\omega$ is a (3×3) diagonal matrix with components of $\omega_{B\_I}^B$ on the diagonal.

$$\delta\omega_{B\_I}^B = \mathbf{b}^B + (\mathbf{D}_\omega\mathbf{s})^B - ([\omega_{B\_I}^B\times]\mathbf{m})^B + \mathbf{w}^B \tag{3.33}$$

The process noise term, $\mathbf{w}^B$, introducing angle random walk is assumed to be a zero-mean, white process with properties established in Equation 3.34. Note that $\delta(\tau)$ is the Dirac delta function.

$$\mathrm{E}\{\mathbf{w}^B(t+\tau)\mathbf{w}^B(t)'\} = \sigma_{rw}^2\mathbf{I}_{3\times 3}\delta(\tau) \tag{3.34}$$

The time derivative of the transformation matrices, $\mathbf{T}_B^I$ and $\hat{\mathbf{T}}_B^I$, are given by Equations 3.35 and 3.36.

$$\dot{\mathbf{T}}_B^I = \mathbf{T}_B^I[\omega_{B\_I}^B\times] \tag{3.35}$$

$$\dot{\hat{\mathbf{T}}}_B^I = \hat{\mathbf{T}}_B^I[(\omega_{B\_I}^B + \delta\omega_{B\_I}^B)\times] \tag{3.36}$$

Taking the time derivative of Equation 3.31 results in Equation 3.37.

$$[\dot{\phi}^B\times] = -\dot{\mathbf{T}}_B^{I\prime}\hat{\mathbf{T}}_B^I - \mathbf{T}_B^{I\prime}\dot{\hat{\mathbf{T}}}_B^I \tag{3.37}$$

Making the proper substitutions to Equation 3.37 and retaining only first order terms

Table 3.6: Kalman Filter States Symbols and Units

Number	State	Description	Units
–	t	Independent variable	s
1-3	ϕ^B	IMU attitude error	radians
4-6	**b**	Gyro bias	radians/s
7-9	**s**	Gyro scale factor	PPM
10-12	**m**	Gyro misalignment	radians

yields Equation 3.38.

$$[\dot{\phi}^B\times] = [\phi^B\times][\omega^B_{B\_I}\times] - [\omega^B_{B\_I}\times][\phi^B\times] - [\delta\omega^B_{B\_I}\times] \tag{3.38}$$

When Equation 3.38 is written in vector form Equation 3.39 results.

$$\dot{\phi}^B = \omega^B_{B\_I} \times \phi^B - \delta\omega^B_{B\_I} \tag{3.39}$$

The purpose of the Kalman filter is to estimate the individual error components in the IMU-provided attitude and angular velocity, and apply these estimates as correction to obtain the system state estimate. The quantities to be estimated (enumerated in Table 3.6) become the states used in the Kalman filter. The units provided in Table 3.6 are the units internal to the Kalman filter. Note that it is important to convert units from the 1-σ values presented in Table 3.1 before implementing the Kalman filter equations.

Define the state vector as a 12-dimensional vector, partitioned into four 3-vectors, as in Equation 3.40.

$$\mathbf{x} = \begin{bmatrix} \phi^B \\ \mathbf{b} \\ \mathbf{s} \\ \mathbf{m} \end{bmatrix} \tag{3.40}$$

Assuming gyro bias, scale factor, and misalignment are modeled as random constants, and using Equations 3.33 and 3.39, the error dynamics can be described by Equation 3.41.

$$\mathbf{F} = \begin{bmatrix} -[\omega\times] & -\mathbf{I} & -\mathbf{D}_\omega & [\omega\times] \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{12\times12}, \quad \mathbf{G} = \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \\ 0 \end{bmatrix}_{12\times3}, \quad \text{and } \mathbf{q}_c = \mathbf{w}^B \tag{3.41}$$

The attitude provided by the star camera is used in conjunction with the angular velocity provided by the IMU to generate a measurement of the first three filter states. The measurement vector is generated in skew-symmetric form in the star

camera frame via Equation 3.42.

$$[\mathbf{z}^{SC}\times] = \mathbf{I} - (\hat{\mathbf{T}}^{I}_{SC\_Star\_Cam})'\hat{\mathbf{T}}^{I}_{SC\_IMU} \tag{3.42}$$

The measurement matrix is used to transform the measurement $\mathbf{z}^{SC}$ into the body frame and is given by Equation 3.43.

$$\mathbf{H}_k = [\mathbf{T}^{B}_{SC}\ 0\ 0\ 0]_{3\times 12} \tag{3.43}$$

Initialization: Notation for the one-sigma error variables used to describe the initialization of the Kalman filter is provided in Table 3.1 and the initial attitude error is defined by Equation 3.44 The initial filter state, $\hat{\mathbf{x}}_{0|0}$, the initial filter error covariance, $\mathbf{P}_0$, the measurement noise covariance, $\mathbf{R}$, and the continuous process noise covariance, $\mathbf{Q}_c$, are given by Equations 3.45, 3.46, 3.47, and 3.48 respectively. Using this implementation of the Kalman filter, only the first term of the Taylor expansion given in Equation 3.16 is nonzero so the discrete process noise covariance is given by Equation 3.49.

$$\text{Initial attitude uncertainty: } \sigma_{ic} = 0.1\text{deg} \tag{3.44}$$

$$\hat{\mathbf{x}}_{0|0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{12\times 1} \tag{3.45}$$

$$\mathbf{P}_0 = \begin{bmatrix} \sigma^2_{ic}\mathbf{I} & 0 & 0 & 0 \\ 0 & \sigma^2_{b}\mathbf{I} & 0 & 0 \\ 0 & 0 & \sigma^2_{sf}\mathbf{I} & 0 \\ 0 & 0 & 0 & \sigma^2_{ma}\mathbf{I} \end{bmatrix}_{12\times 12} \tag{3.46}$$

$$\mathbf{R} = \begin{bmatrix} \sigma^2_{sc\_r} & 0 & 0 \\ 0 & \sigma^2_{sc\_p,y} & 0 \\ 0 & 0 & \sigma^2_{sc\_p,y} \end{bmatrix} \tag{3.47}$$

$$\mathbf{Q}_c = \sigma^2_{rw}\mathbf{I}_{3\times 3} \tag{3.48}$$

$$\mathbf{Q}_k \approx \mathbf{M}\Delta t \tag{3.49}$$

Propagation, Measurement, and Reset: The propagation process occurs just as described in Equations 3.20 and 3.21 at a rate equivalent to the output of the IMU, which in this case is 100Hz. The measurement process proceeds as prescribed by Equations 3.22, 3.23, 3.24 at the star camera update rate, which is nominally one update every five seconds. The reset equations require that additional system level variables be defined to match the filter state variables 4-12. These variables, ${}_s\mathbf{b}^B$, ${}_s\mathbf{s}^B$, and ${}_s\mathbf{m}^B$, are initialized with 3×1 vectors of zeros, and will store the cumulative effect of the errors used to correct the angular velocity values from the IMU. At each

measurement/reset time the system level variables are updated via Equation 3.50.

$$\begin{aligned} {}_s\mathbf{b}^B &\leftarrow {}_s\mathbf{b}^B + \mathbf{b}^B \\ {}_s\mathbf{s}^B &\leftarrow {}_s\mathbf{s}^B + \mathbf{s}^B \\ {}_s\mathbf{m}^B &\leftarrow {}_s\mathbf{m}^B + \mathbf{m}^B \end{aligned} \tag{3.50}$$

The angular velocity measurements from the IMU are corrected using the system level variables as described in Equation 3.51.

$$\bar{\omega}^B_{B\_I}(k) = \hat{\omega}^B_{B\_I}(k) - {}_s\mathbf{b}^B - \mathbf{D}_{\omega_k s}\mathbf{s}^B + [\omega^B_{B\_I}(k)\times]{}_s\mathbf{m}^B \tag{3.51}$$

The attitude estimate is reset using filter states 1-3, where the current attitude is transformed by the estimated attitude error and used as the initial condition for a system level quaternion integration calculation (see Equation 3.52). A quaternion integration calculation employing Equation 3.2 is required at the system level (exterior to the filter) because the IMU supplies only angular velocity measurements. The process of using the corrected attitude as an initial condition for quaternion integration works to accumulate the attitude error corrections in the system level description of the attitude.

$$\hat{\mathbf{T}}^I_B = \hat{\mathbf{T}}^I_B \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_1^B & \sin\phi_1^B \\ 0 & -\sin\phi_1^B & \cos\phi_1^B \end{bmatrix} \begin{bmatrix} \cos\phi_2^B & 0 & -\sin\phi_2^B \\ 0 & 1 & 0 \\ \sin\phi_2^B & 0 & \cos\phi_2^B \end{bmatrix} \begin{bmatrix} \cos\phi_3^B & \sin\phi_3^B & 0 \\ -\sin\phi_3^B & \cos\phi_3^B & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.52}$$

Once the system level variables are updated, the filter state is reset. Because all of the filter states are used to reset system level variables, the matrix $\mathbf{A}_2$ in Equation 3.25 is the 12 identity matrix, which resets the filter state to the initial condition given in Equation 3.45.

Navigation results

A simple trajectory is run in simulation to produce typical navigation system results. The true angular velocity profile and the associated quaternion values are illustrated in Figure 3-4. The error values used in the environment are provided in the Table 3.7 along with the values the filter converged to by the end of the simple trajectory. Figure 3-5 displays the difference between the true and estimated quaternion values of the satellite attitude and the angular velocity error throughout the trajectory. Finally, Figure 3-6 presents the standard deviations of the attitude estimation errors, the values of which correspond to the square root of the first three diagonal elements of the filter's state covariance matrix.

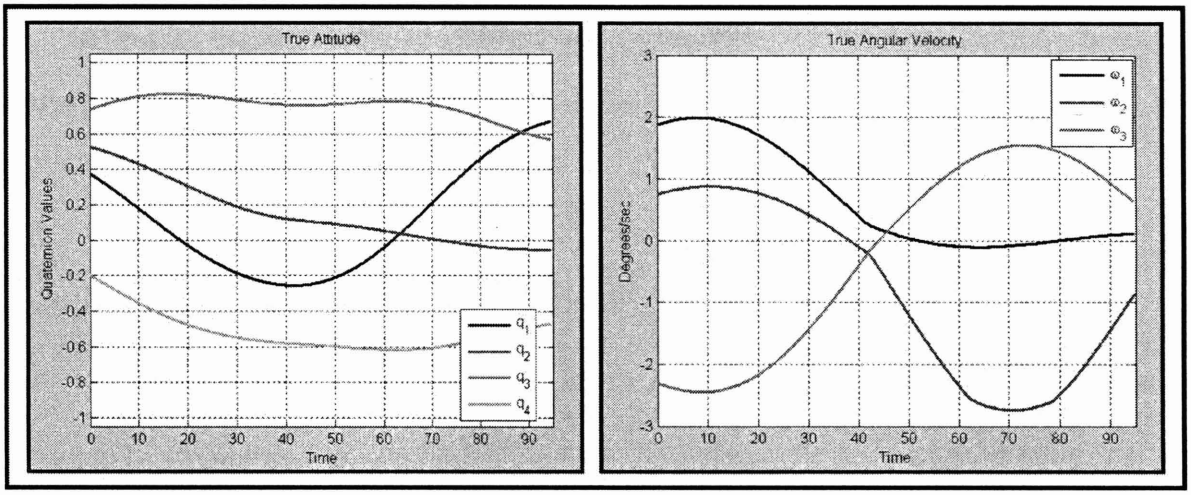

Figure 3-4: The angular velocity and quaternion values of a simple trajectory used to produce navigation system results.

Table 3.7: Navigation Environment and Filter Error Values

Error		**Environment**	**Filter**	**Units**
Bias:	x	1.0	0.63	deg/hr
	y	0.7	-0.26	
	z	-1.0	-1.08	
Scale Factor:	x	100	13	PPM
	y	70	-9	
	z	-100	-7	
Misalignment:	x	0.1	0.006	micro-radians
	y	-0.1	0.004	
	z	0.15	-0.010	

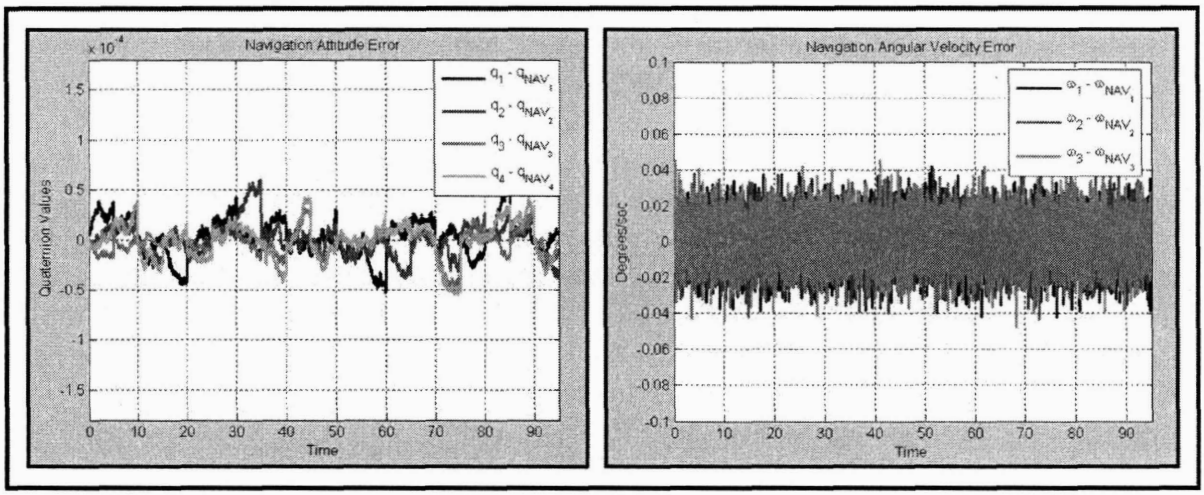

Figure 3-5: The angular velocity and quaternion errors

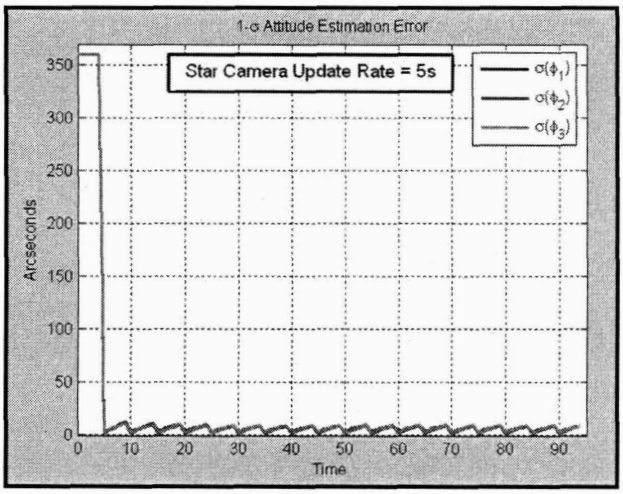

Figure 3-6: The standard deviation of the attitude estimation error (per axis)

3.2.2 Predicted Translational State Data and Noise Model

For the purposes of simulation, the true translational states of the satellites and the object were generated using Satellite Tool Kit. The covariance of the satellite's translational state prediction errors is determined using the GPS errors listed in Table 3.1 via equation 3.53. The covariance of the object is chosen to have large uncertainties in both position and velocity. The covariance of the object's translational state is given in 3.54. Every time the navigation system receives a request for an initial translational state to propagate forward in time, it randomly selects the initial state according to the statistics provided by the estimation error covariance. This method of selecting an error-prone initial state serves as the noise model used in simulation. The estimated initial state and the error covariance are then propagated forward in time using a simple gravitational model. For this experiment it is assumed that the satellites and the object are non-thrusting bodies, and the dynamics are only influenced by the force of gravity.

$$\mathbf{P}_{sat} = \begin{bmatrix} \sigma_{Sp}^2 \mathbf{I}_{3\times3} & \mathbf{0} \\ \mathbf{0} & \sigma_{Sv}^2 \mathbf{I}_{3\times3} \end{bmatrix}_{6\times6} \tag{3.53}$$

$$\mathbf{P}_{obj} = \begin{bmatrix} \sigma_{Op}^2 \mathbf{I}_{3\times3} & \mathbf{0} \\ \mathbf{0} & \sigma_{Ov}^2 \mathbf{I}_{3\times3} \end{bmatrix}_{6\times6} \tag{3.54}$$

The equation used to propagate translational states is provided in Equation 3.55, where $\mathbf{r}$ is the position in an Earth centered inertial reference frame, $\mathbf{v}$ is the velocity in the same inertial frame, G is the gravitational constant, and M_e is the mass of the Earth.

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \end{bmatrix} \\ \dot{\mathbf{r}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \frac{GM_e}{||\mathbf{r}||^3}\mathbf{r} \end{aligned} \tag{3.55}$$

The equation used to propagate the translational state error can be derived from Equation 3.55. Let $\dot{\mathbf{x}} = F(\mathbf{x})$ represent Equation 3.55, then substitute $\hat{\mathbf{x}} + \delta\mathbf{x}$ for $\mathbf{x}$ where $\delta\mathbf{x}$ represents the error in estimate $\hat{\mathbf{x}}$. Retaining only the first order term of a Taylor series expansion yields Equation 3.56.

$$\dot{\hat{\mathbf{x}}} + \delta\dot{\mathbf{x}} = F(\hat{\mathbf{x}}) + \nabla F|_{\hat{\mathbf{x}}}\delta\mathbf{x} \tag{3.56}$$

The error dynamics are then described by Equation 3.57.

$$\begin{aligned} \delta\dot{\mathbf{x}} &= \nabla F|_{\hat{\mathbf{x}}}\delta\mathbf{x} \\ \text{where } \nabla F|_{\hat{\mathbf{x}}} &= \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \frac{-GM_e}{||\hat{\mathbf{r}}||^3}(\mathbf{I} - 3\frac{\hat{\mathbf{r}}}{||\hat{\mathbf{r}}||^2}\frac{\hat{\mathbf{r}}'}{||\hat{\mathbf{r}}||^2}) & \mathbf{0} \end{bmatrix} \end{aligned} \tag{3.57}$$

Notice how the error dynamics are a function of the time dependent variable $\mathbf{r}$. By choosing a small enough time interval Δt so that $\nabla F|_{\hat{\mathbf{x}}}$ can be approximated by a constant matrix between times t and $t + \Delta t$ the discrete state transition matrix, Φ, can be described by Equation 3.58.

$$\Phi_k = e^{[\nabla F|_{\hat{\mathbf{x}}_k}]\Delta t} \tag{3.58}$$

The estimation error covariance matrix, P, can then be propagated using the state transition matrix via Equation 3.59

$$P_{k+1} = \Phi_k P_k \Phi_k' \tag{3.59}$$

It is acknowledged that very simple noise model is used to derive translational state predictions for the simulation. Within a real system it is likely that an Extended Kalman filter would be used to combine measurements, resulting in refined initial state estimates. Furthermore, the initial state estimates would likely be entered into a realistic orbit propagator that can account for a non-uniform gravitational field, atmospheric drag, etc. Nevertheless, the method outlined above is sufficient for modeling objects in Keplarian orbits and over short durations. Figure 3-7 illustrates the error growth associated with predicting the future translational states of the satellite and the object through the use of covariance propagation. In Figure 3-7 the standard deviation with the largest value of the three position components is selected for each plot.

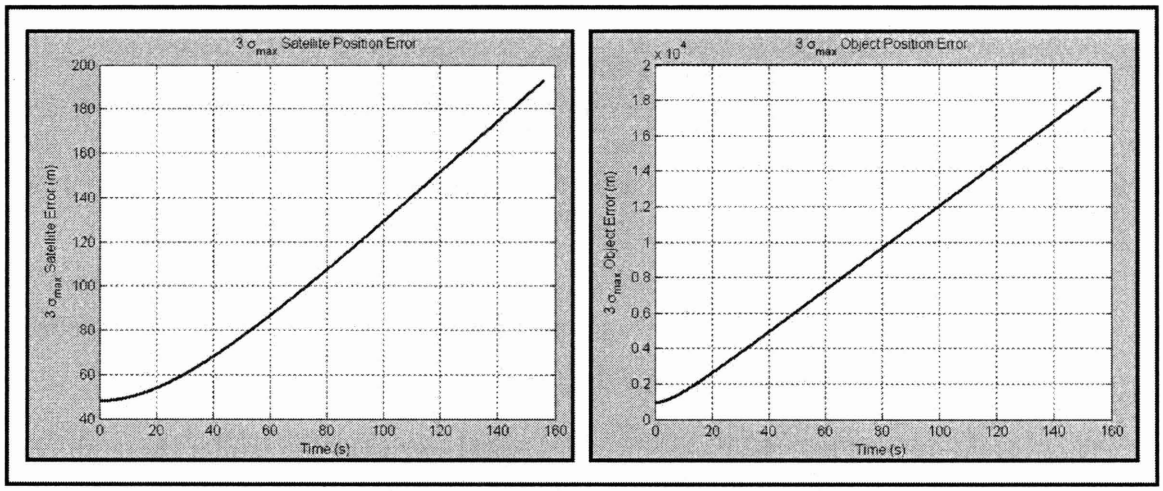

Figure 3-7: The translational error propagation of the satellite and the object tracked by the satellite.

3.3 Controller

A simple feedforward feedback trajectory following controller is implemented to execute the trajectory provided by the planner. The planner supplies feedforward torques to maneuver the satellite through the planned trajectory. The feedforward torques are based upon an error-prone model of the satellite's dynamics which is internal to the planner. Without feedback corrections, the modeling errors contained within the feedforward torques would cause a growing deviation between the satellite's trajectory and the intended trajectory. So to keep the satellite's trajectory closely matched to the planned trajectory, feedback corrections are derived from the difference between the rotational state estimates provided by the navigation system and the corresponding values in the planned trajectory. In this section a brief description of the controller is provided, followed by a derivation of an associated Lyaponov function. A brief discussion about errors associated with the controller is also included and the section concludes with some results illustrating the controller's performance.

The controller drives the satellite along the planned trajectory through the use of feedback corrections. The control actions referenced in Equation 3.1 are specified by the control law provided in Equation 3.60. Note that ω_p represents the planned angular velocity of the body frame with respect to the inertial frame formulated in the body frame, $\mathbf{q}_p$ represents the planned body to inertial satellite attitude, and the subscripts v and s respectively denote the vector and scalar portions of a quaternion.

$$M(\mathbf{u}^B) = \hat{\omega}^B_{B\_I} \times \hat{\mathbf{J}}\hat{\omega}^B_{B\_I} + \hat{\mathbf{J}}\dot{\omega}_p - k_d\hat{\mathbf{J}}\omega_e - k_p\hat{\mathbf{J}}\mathbf{q}_{e_v}q_{e_s} \tag{3.60}$$

$$\begin{aligned} \text{where} \quad \omega_e &= \omega_p - \hat{\omega}^B_{B\_I} \\ \text{and} \quad \mathbf{q}_e &= \mathbf{q}_p^{-1} \otimes \hat{\mathbf{q}}^I_B \end{aligned} \tag{3.61}$$

Assuming the term $\hat{\omega}^B_{B\_I} \times \hat{\mathbf{J}}\hat{\omega}^B_{B\_I}$ successfully negates the term capturing the gyroscopic moments in Equation 3.1, $-\omega^B_{B\_I} \times \mathbf{J}\omega^B_{B\_I}$, with negligible error and that moments applied to the satellite by environmental effects, $M(\mathbf{v}^B)$, are also negligible then the angular velocity error dynamics can be described by Equation 3.62.

$$\begin{aligned} \hat{\mathbf{J}}\dot{\omega}_e &= -k_d\hat{\mathbf{J}}\omega_e - k_p\hat{\mathbf{J}}\mathbf{q}_{e_v}q_{e_s} \\ \text{where} \quad \hat{\mathbf{J}}\dot{\omega}_e &\approx \mathbf{J}\dot{\omega}^B_{B\_I} - \hat{\mathbf{J}}\dot{\omega}_p \end{aligned} \tag{3.62}$$

The error dynamics are used to prove the stability of the controller through the use of an associated Lyaponov function. A suitable choice of Lyaponov function for this controller is given by Equation 3.63.

$$\mathcal{L} = \frac{1}{2}\omega'_e\omega_e + k_p\mathbf{q}'_{e_v}\mathbf{q}_{e_v} \tag{3.63}$$

The derivation of the time derivative of the Lyaponov function is given by Equation 3.64. It can be seen that the time derivative of the Lyaponov function is negative

semi-definite indicating that the tracking controller is globally asymptotically stable.

$$\begin{aligned}\dot{\mathcal{L}} &= \frac{1}{2}\omega_e'\dot{\omega}_e + k_p\mathbf{q}_{e_v}'\dot{\mathbf{q}}_{e_v} \\ &= \frac{1}{2}\omega_e'(-k_d\omega_e - k_p\mathbf{q}_{e_v}q_{e_s}) + \ldots \\ &\quad k_p\mathbf{q}_{e_v}'(\frac{1}{2}q_{e_s}\omega_e + \frac{1}{2}\mathbf{q}_{e_v}\times\omega_e) \\ &= -\frac{1}{2}k_d\omega_e'\omega_e \end{aligned} \tag{3.64}$$

There are several sources of error within the controller that prevent it from perfectly executing the planned trajectory. First, there may be mismodeled dynamics within the planner, which may include error prone estimates of hardware parameters, misaligned and nonorthogonal reference frames, and control actuator dynamics. These errors result in error prone feedforward control actions, which in Equation 3.60 is represented by the term $\hat{\mathbf{J}}\omega_p$. Secondly, the feedback control actions are based upon error prone estimates of the system's state provided by the navigation function. The feedback control actions will work to drive the navigation system's estimate of the system state toward the desired trajectory which may be different then driving the actual system to the desired trajectory.

A simple trajectory is developed in simulation to demonstrate the performance of the controller. The planned angular velocity and attitude profile is shown in Figure 3-8. Noisy angular velocity and attitude estimates provided by the navigation function are illustrated in Figure 3-9. The true and estimated moment of inertia matrices are given in Equation 3.65. The control gains, k_d and k_p are equal to 1 and 0.2 respectively. The simulation is initiated with the angular velocity and quaternion errors given by Equation 3.66. The angular velocity and quaternion errors are presented in Figure 3-10. Finally, the feedforward, feedback, and total control torques are shown in Figure 3-11.

$$\mathbf{J} = \begin{bmatrix} 120 & 0 & 0 \\ 0 & 125 & 0 \\ 0 & 0 & 115 \end{bmatrix} \qquad \hat{\mathbf{J}} = \begin{bmatrix} 120.370 & 1.135 & 2.289 \\ 0.457 & 124.386 & 2.844 \\ 1.659 & 0.894 & 114.951 \end{bmatrix} \tag{3.65}$$

$$\omega_e = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{q}_e = \begin{bmatrix} 0.9853 \\ 0.0985 \\ 0.0985 \\ 0.0985 \end{bmatrix} \tag{3.66}$$

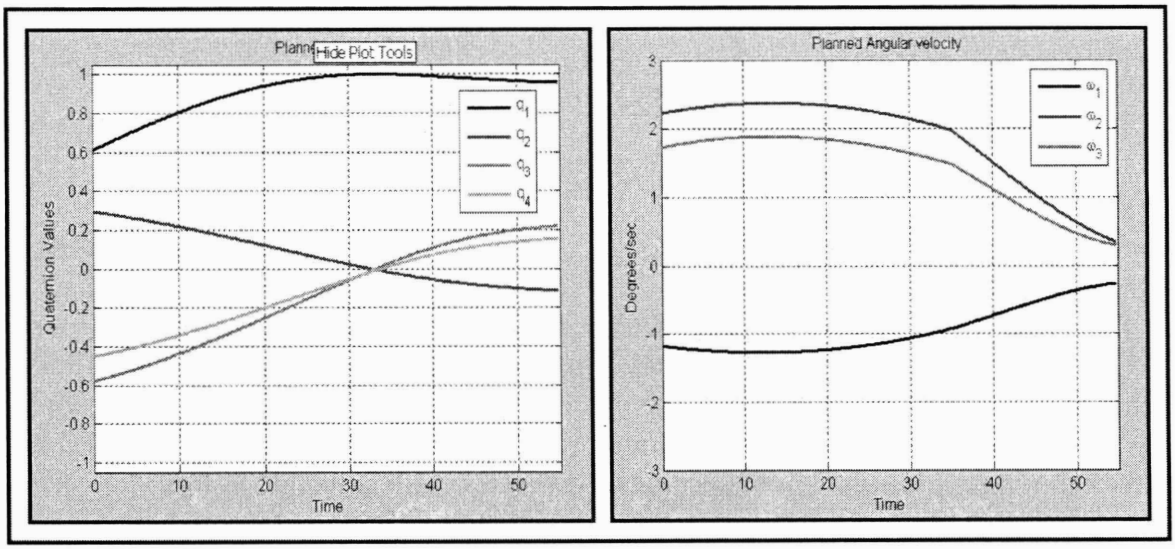

Figure 3-8: The planned trajectory

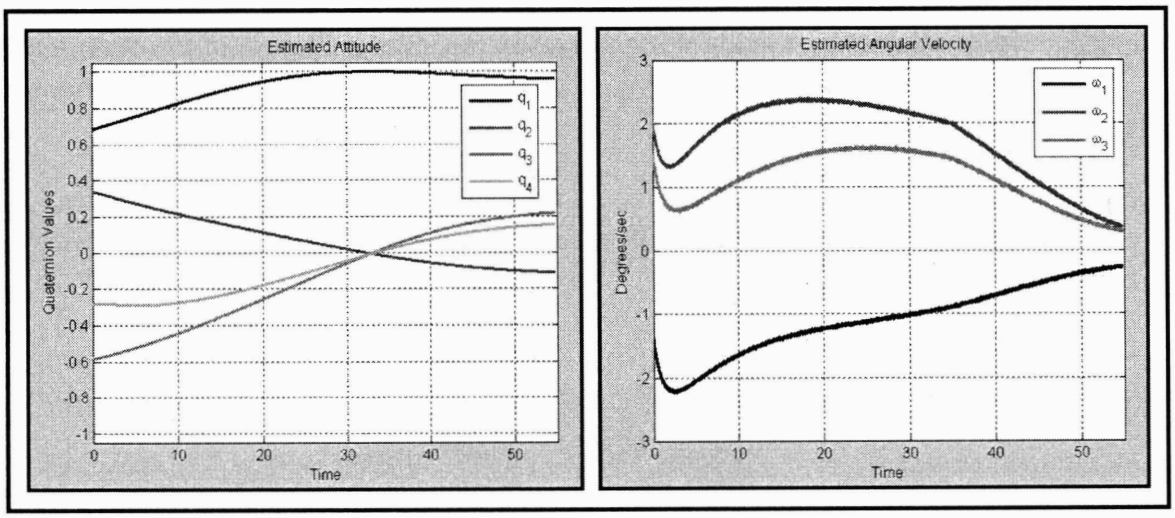

Figure 3-9: The trajectory estimated by the navigations system

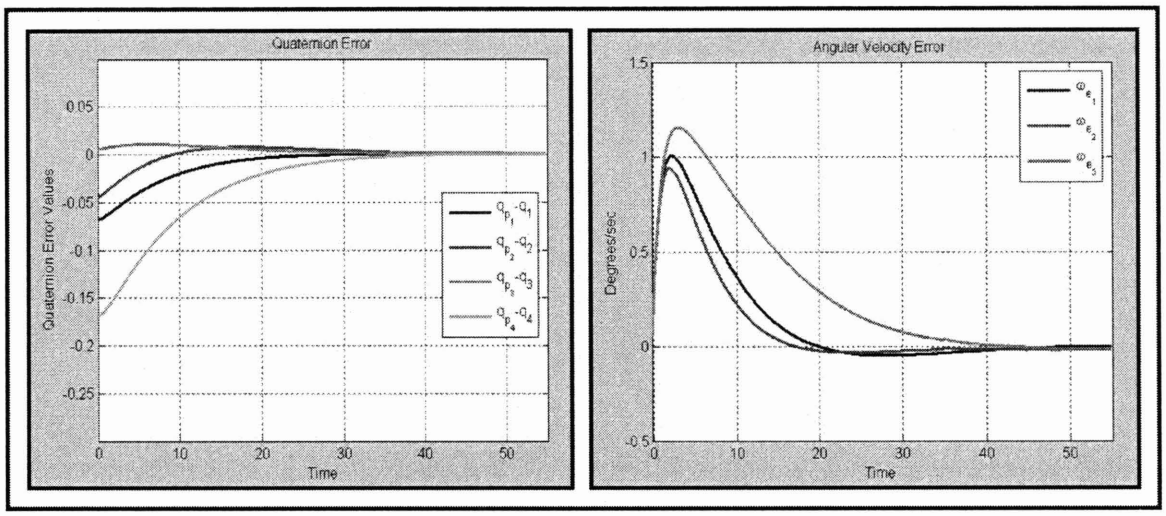

Figure 3-10: The quaternion and angular velocity errors

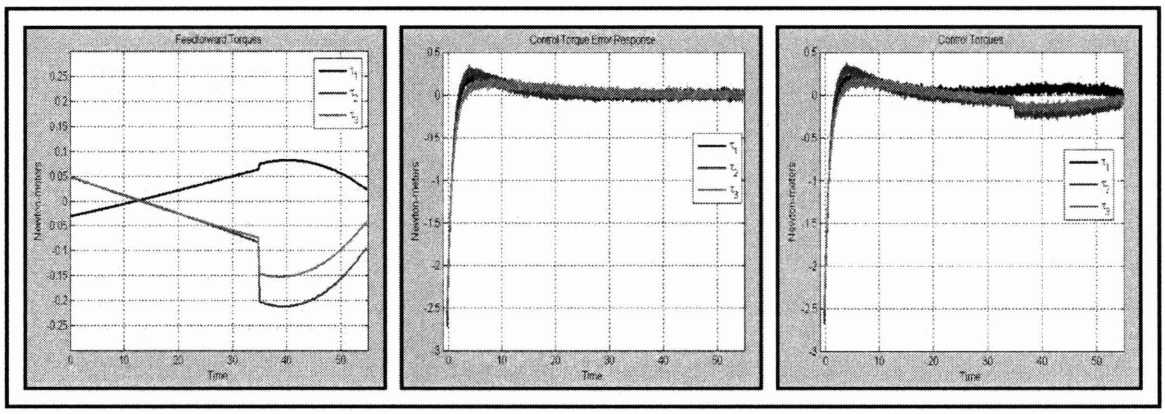

Figure 3-11: The feedforward, feedback and total control torques

Chapter 4

Outer-loop Description

The outer planning loop contains the two main contributions of this thesis. The first contribution resides in the implementation of the RRHC, a robust receding horizon planner. This task is accomplished through the use of the preplanner, which facilitates closing the loop around an open-loop planning algorithm. Expected errors with known bounded norms are used by the preplanner to produce constraint definitions supplied to the planner to generate a trajectory robust to those errors. The first section of the chapter will present the concept of the preplanning function and describe the specific implementation used for this experiment. The second main contribution is found in the RSTP which is a new algorithm that extends the RRT concept to problems with dynamic environments. The RSTP algorithm will be discussed in Section 2 and compared to Frazolli's algorithm, which was introduced in Chapter 2. Finally, the specific application of the new algorithm will be presented.

4.1 Preplanner

The preplanner is critical for closing the outer planning loop around an open-loop planning algorithm. In a general sense, the preplanner is the set of calculations required to implement an open-loop planner as a robust receding horizon planner. Many approaches may be taken with a number of different open-loop planners to accomplish this task. This section will present an approach and discuss a number of the issues that must be considered when developing a preplanning function.

4.1.1 Purposes and Elements of Preplanner

The preplanner has two primary purposes; the first is to provide conditions for the stability of the outer, planning loop, and the second is to provide the planning function with the input required to generate the receding horizon trajectory segments. The output of the preplanner is characterized by four elements which affect both the stability of the outer loop and the execution of the planning function. The planner output elements include: constraint size definitions, function call times, initial planning conditions, and end-plan conditions. The constraint size definitions are required

by the collision checking component of the planner and are based on calculations that ensure that the plan will be robust to errors while also ensuring that the problem is not too tightly constrained. The function call times are the times at which the pre-planning and planning functions are called to generate the next trajectory segment. The initial planning conditions establish the root of the search tree to be generated by the planner. Lastly, the end-plan conditions define the circumstances under which the planner terminates and returns a new trajectory.

Different initial planning and end-plan conditions are required by different types of problems. A tracking problem may not have a specific end goal, but only the requirement that the constraints of the problem are satisfied for a certain amount of time. On the other hand, goal-oriented problems require the vehicle to obtain a certain configuration at the end of the planned trajectory without strict requirements on the time that the goal configuration is achieved. Intercept problems demand that the configuration of the vehicle at a given time match the time-varying configuration of the goal.

4.1.2 Stability Considerations for Outer, Planning Loop

Projecting Initial Conditions

The initial planning conditions usually consist of an initial state and time. Within the receding horizon implementation, the initial state and time sent to the planner must be the projected state and time of the vehicle at some point in the future beyond the time it takes for the planner to compute a solution. For example, if the planner takes 10 seconds to generate a plan, then the initial planning time must be a time beyond 10 seconds, and the initial state must correspond to the projected vehicle state at that future time. In this way the computed plan is ready when the vehicle arrives at or near the initial state used in the planner.

Determination of End-Plan Conditions

A method of determining end-plan conditions must be devised such that a series of trajectory segments will maneuver the vehicle or system to the goal state or configuration. For the tracking problem, because the goal is not a specific state or configuration but only a span of time for which a feasible path must be generated, a valid trajectory segment consists of a branch of the search tree that extends to the end-plan time. In order to prevent the receding horizon planner from providing a trajectory segment that leads to an entrapment behind constraints, a plan can be attempted to the final goal with the understanding that only the initial segment of the solution is robust to errors and uncertainties. Only the robust (finite-time) portion of the trajectory is used to drive the system, while the remainder of the trajectory (infinite time) is calculated to guarantee the stability of the solution. This approach will be referred to as the infinite horizon approach, and is similar to some approaches taken in model predictive control (MPC), such as the MPC and Control Lyuapunov function methods of Hauser and Jadbabaie [24] or the full horizon MPC approach discussed in [43].

For the infinite horizon approach. the end-plan conditions passed to the planner will always be the final goal.

The infinite horizon approach for constructing trajectory segments has two additional benefits besides preventing the generation of segments that become entrapped by constraints. For goal-oriented and intercept problems, infinite horizon planners are complete. Additionally, if for some reason the planner is unable to generate a robust trajectory after an initial trajectory segment has been generated, the remaining open-loop plan is always available as a fall back option.

Correlation of Constraint Boundaries and Error Growth

A distinction can be made between the future span of time for which the planner generates a trajectory segment, and the time in which the system executes the planned trajectory. The former is referred to as prediction time and the latter is referred to as execution time. Many robotic vehicles are built with sensors which inhibit the growth of uncertainty in the vehicle's state by taking measurements throughout the execution time. These sensor updates allow the vehicle to closely track a planned trajectory. However, the information from these updates is not available to the planner because the sensor updates do not occur until after the formation of the plan. So in prediction time, the uncertainty in the vehicle's state will inevitably increase. For problems that require a plan that is robust to errors and uncertainty that grow in prediction time, the constraint boundary definitions and the planning time horizon will be correlated. The constraint boundaries must be enlarged to handle the effect of the errors at the time the errors are largest, which for errors that grow in prediction time occurs at the end of the planning time horizon. If the planning time horizon is far in the future or the errors grow very quickly in prediction time, then the constraint boundary enlargement will quickly absorb the existing free-space. In order to prevent the planning problem from becoming too tightly constrained, an upper bound should be established on the degree to which the constraints are enlarged. This upper bound on constraint boundary enlargement subsequently sets an upper bound on the planning horizon time.

The function call times are established to ensure that adequate planning time is reserved for the computation of the next trajectory segment. Because randomized trajectory planners cannot guarantee the convergence to a solution within a set amount of time, the time allowed for generating a trajectory segment must be established through a statistical measure of computation times. From Monte Carlo simulation runs, in which a large number of trajectory segments are generated under many different simulated conditions and corresponding computation times recorded, one can experimentally determine a computation time that will capture a high percentage of the sample simulation runs. If the sample of simulation runs is diverse enough, the percentage of runs successfully resulting in a trajectory segment within the computation time chosen can be considered the algorithms reliability for the allotted computation time. In the end, the choice of computation run time and algorithm reliability is problem specific and must be resolved as a function of the problem and domain.

Minimum Trajectory Segment Time

In order to ensure the stability of the outer planning loop, the allotted computation time must not exceed the time spanned by a trajectory segment. In other words, the current plan must extend far enough into the future to allow a new plan to be generated. This stability requirement establishes a ceiling on the function call time. The planner must begin planning no later than $t_{EOS} - t_{comp}$, where t_{comp} is the allotted computation time and t_{EOS} is the time at which the current trajectory segment ends. Furthermore, the trajectory to be planned must not end before $t_{EOS} + t_{comp}$.

The preplanner may run cases where, either because of rapid error growth or long computation times, a robust trajectory segment that spans the computation time cannot be computed without imposing constraint boundaries that dominate the free-space. In these cases, compromises must be made in order to generate a robust plan. One approach would be to drop any nonessential constraints from the problem, thereby opening up the free-space. Another approach would be to examine the algorithm to find ways of speeding up the convergence to a solution. For example, if a smoothing function is used in conjunction with the planning function, then the call to the smoothing function may be omitted and the computation time of the smoothing function saved. Although, if neither option is available, then a robust plan may have to be forfeited.

Function Call Times

It is understood that the navigation system is continuously providing the most accurate system and environment information available. Formulating a plan earlier than necessary, would mean that the latest navigation information would not be used in the generation of the plan. In other words, the propagation time, and the associated error growth begin at the time the function is called, so the function should be called with only enough time to finish computations before executing the plan. This point is illustrated in Figure 4-1 where t_{fc} is the time of the function call, t_{BOP} is the time that marks the initial point of plan execution, and t_{EOP} is the time that marks the end of the engagement.

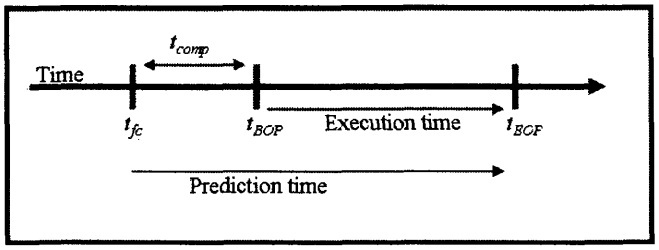

Figure 4-1: A timeline illustrating function call, prediction, and execution times.

Smooth Boundary Transitions

When constraint boundary definitions are based on error-prone estimates that are updated for each planning segment, there must be a smooth transition from the old set of boundary definitions to the updated definitions. This point is best illustrated by an example. Figure 4-2 was developed by running the third engagement scenario in simulation. The thick dotted line represents the true angle between the line-of-sight vector to the object and the primary camera bore-sight vector, while the thin solid line represents the angle between the anticipated line-of-sight vector to the object and the primary camera bore-sight vector. A step in the thin solid line occurs at the times when a new plan is formulated (approximately 70 and 170 seconds). The step is created when the primary constraint boundary is redefined using updated translational state data. The step transition of the boundary constraints from one planning segment to the next could cause an infeasible starting point for one of the planning segments, and ultimately a breakdown of the planning function. Because the constraint boundary definitions are generated before the plan is developed, the problem is easily overcome by making the transition from the old constraint boundary definitions to the new constraint boundary definitions smooth. The transition must be gradual enough that the system can dynamically adjust to the new constraint boundaries should the system reside near an artificial constraint boundary at the beginning of a new planning segment. In order to make the transition smooth in simulation, the first five seconds of the constraint boundaries for each planned segment are the result of a weighted interpolation between the old constraint boundary and the new constraint boundary.

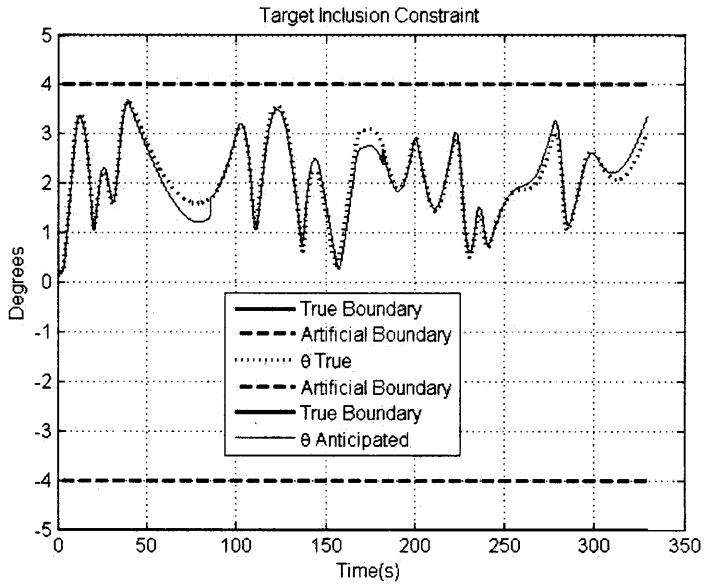

Figure 4-2: Primary camera constraint satisfaction for Scenario 3.

Summary

To summarize, the preplanning function essentially bridges the gap between information supplied by the navigation system and the planning function. The information about the system and the environment is used to compute the initial planning conditions, end-plan conditions, and constraint boundary definitions required by the planning function. Furthermore, the function call times, constraint boundary definitions and end-plan conditions are chosen to facilitate the stable execution of the outer planning loop.

4.1.3 Application of Preplanner

As discussed in the previous section, the preplanner must identify the constraint size definitions, the function call times, the initial planning conditions and the end-plan conditions. For the preplanner implemented in this experiment, constraint size definitions are made for the star camera constraints, the communications antenna constraint, and the primary camera constraint. This section presents the approach used to generate constraint size definitions and explains the initial planning and end-plan conditions used. Because the implemented preplanning and planning functions are not optimized for the shortest computation times, function call times for this experiment are based upon simulated computation times.

Constraint Size Definitions

Recall that a plan is made robust to errors by artificially enlarging the boundaries of the constraints beyond the effect of the errors will have on the output of the system. In order to determine the degree of artificial constraint boundary enlargement, a preplanning function uses error statistics and system parameters to estimate the upper bound of the effect of the errors on the output of the system. The upper bound of some error sources may be analytically calculated; however, often the complexity of the system precludes a tractable analytic solution. If an analytic solution is not available, the upper bound of an error source can be estimated via the use of results from a linearized covariance analysis or Monte Carlo simulation.

Within the simulated OTE system there are two main errors for which upper bounds will be estimated by a preplanning function. The two error sources are pointing errors generated when propagating estimated trajectories that define constraint boundaries (propagation error); and pointing errors caused by a controller executing with error prone navigation information and mismodeled dynamics (controller error). An upper bound on controller error is determined empirically, while the upper bound on propagation error is estimated by the results of a linearized covariance analysis.

The preplanner implemented in this experiment sets a maximum constraint enlargement to 1 degree in order to prevent the resulting path planning problems to be too tightly constrained. The preplanner calculates the sum of the pointing errors generated by the different sources and determines the time when the cumulative effect of the pointing errors exceeds the maximum constraint enlargement value, which is

referred to as the cutoff time. The cutoff time marks the end of the next robust trajectory planning segment. The next function call time is determined by subtracting the computation time from the cutoff time.

Controller Error Calibration

The two main contributing factors to controller error include the error-prone feedback signal from the navigation system and mismodeled dynamics within the planning function. The nominal torque values produced by the planning function are used in the feedforward, feed-back controller described in Section 3.3. Estimated angular velocities are used to both provide feedback on the reference values provided by the planner and to calculate the torques required to cancel the gyroscopic moments as described in Equation 3.60. As a result, the performance of the controller is dependent upon the quality of the navigation estimate of the satellite's rotational state. The navigation estimate is most affected by the update rate of the star camera, so the controller's performance is calibrated using several different star camera update rates in the navigation function; namely, 0.5Hz, 0.2Hz, and 0.1Hz.

The effect of mismodeled dynamics is captured by using an error prone moment of inertia matrix. Errors in the measured moment of inertia values result in controller error for two reasons: one, the estimate of the moment of inertia is used to cancel out gyroscopic forces as described in Equation 3.60, and two, the execution of feedforward torque values will not result in the angular velocities anticipated by the planner. To introduce error into the moment of inertia matrix, values of the moment of inertia matrix are chosen randomly from a normal distribution where the true value is used as the mean and the variance is set to a maximum of 1% of the true value.

The controller error is determined empirically though the analysis of a set of simulated test cases. For each test case, a trajectory is generated at random to contain angular velocities and accelerations that approach the limit of the satellite's capabilities. Each trajectory is then executed several times over with various combinations of moment-of-inertia estimates and star camera update rates. If the controller operates perfectly, using perfect navigation information and a model perfectly matching the environment, then the rotational trajectory taken by the satellite would exactly match the reference trajectory fed into the controller. Therefore, the quaternion values mapping out the reference trajectory are regarded as the true values, while the quaternions returned by the simulated satellite attitude dynamics acting under the control law specified in Section 3.3 are regarded as error prone values. The vector representing the bore-sight of the primary camera in the body frame is transformed by both the body-to-inertial quaternion values in the reference trajectory and the quaternion values produced by the simulated satellite dynamics which represent the true satellite attitude. The difference between the two different inertial representations of the same vector is used as a measure of the controller error. Equation 4.1 presents the calculation used to determine the controller error, where $\mathbf{q}_{ref}$ is the reference body to inertial quaternion supplied to the controller, $\mathbf{q}_B^I$ represents the actual attitude of the satellite after the control actions have been applied to the system, $\theta_{v_{pc}}$ represents the pointing error induced by the controller, and $\mathbf{v}_{pc}$ represents the

bore-sight vector of the primary camera. Figure 4-3 illustrates the pointing error for a series of test cases. The controller error does not tend to grow in time and based upon the results, a constant upper bound can be set at 0.07 degrees.

$$\begin{aligned} \mathbf{v}_{pc}^{I} &= \mathbf{q}_{B_{ref}}^{I} \otimes \begin{bmatrix} 0 \\ \mathbf{v}_{pc}^{B} \end{bmatrix} \otimes \mathbf{q}_{B_{ref}}^{I^{-1}} \\ \hat{\mathbf{v}}_{pc}^{I} &= \mathbf{q}_{B}^{I} \otimes \begin{bmatrix} 0 \\ \mathbf{v}_{pc}^{B} \end{bmatrix} \otimes \mathbf{q}_{B}^{I^{-1}} \\ \theta_{\mathbf{v}_{pc}} &\approx \|\mathbf{v}_{pc} - \hat{\mathbf{v}}_{pc}\| \end{aligned} \tag{4.1}$$

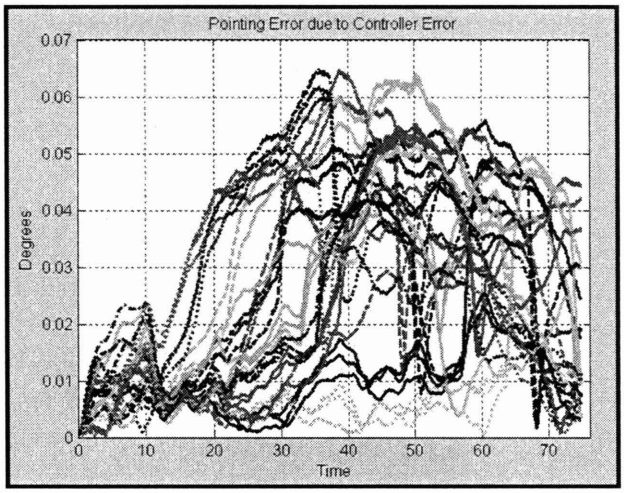

Figure 4-3: The magnitude of the pointing error induced by controller errors.

Translational State Propagation Error Calibration

While the controller error prevents the plan from being perfectly executed, the propagation errors prevent the formulation of a perfect plan. The constraint boundaries for the attitude trajectory planning problem are defined by the propagated translational position of the satellites and the object. If the constraint boundaries formed within the plan do not match the actual constraint boundaries then a constraint violation could occur. Therefore, the effect of the propagation errors must also be folded into the constraint size definitions in order to construct a robust plan.

The constraint definitions for the star camera are formed by using the propagated translational state of the satellite and the ephemerides of the Sun, Earth and Moon. At discrete intervals along the projected path the position of the satellite is subtracted from the position of the Sun, Earth, and Moon. The resulting vectors are normalized to provide exclusion cone defining vectors. Because some error exists in the propagated translational state of the satellite, there will be some error in the resulting exclusion cone defining vectors; however, because the size of the propagated

state error is extremely small compared to the distances between the satellite, and the Sun, Earth, and Moon, the error in the exclusion cone defining vectors can be safely neglected in this experiment.

The constraint definitions for the communications antenna are formed by using the propagated translational states of the satellites. To form the constraint definitions of the first satellite's communications antenna, the position of the first satellite is subtracted from the position of the second satellite at discrete intervals along the propagated paths of the satellites. The resulting vectors are normalized to be used as exclusion cone defining vectors. Exclusion cone defining vector errors induced by errors in the propagated translational states of the satellites may become sizable if the plan extends far into the future; however, for the time scales used in this experiment, the communications antenna exclusion cone errors are insignificant, and therefore neglected.

The inclusion cone defining vector for the primary camera is generated by subtracting the propagated translational state of the satellite from the propagated translational state of the object and normalizing the resulting vector. Uncertainties in both the propagated satellite position and the object's propagated position generate a significant amount of error in the inclusion cone defining vector. An upper bound to the error associated with the inclusion cone defining vector is estimated by propagating the covariance of the estimation errors supplied by the Navigation system via Equation 3.59. The square root of each of the first three diagonal elements of the satellite's (object's) translational state covariance matrix provides the standard deviations of the satellite's (object's) position. To obtain a conservative estimate on the maximum deviation of the satellite (object) from its estimated position, the largest standard deviation among the three axes is selected and multiplied by three. The resulting value is referred to as the probable error radius. The maximum pointing error in the unit vector pointing from the satellite to the object is calculated using the probable error radius of the satellite, r, and the target, R. In Figure 4-4 it can be seen that the maximum pointing error, α, can be computed from simple geometric relationships. The shaded circles represent the probable error radius of the satellite and the object. Equation 4.2 provides the approximate value of the propagation pointing error. Finally, Figure 4-5 displays the propagation error growth in time using the initial satellite covariance and object covariance provided in Equations 3.53 and 3.54.

$$\alpha \approx \tan\alpha = \frac{R+r}{\sqrt{L^2-(R+r)^2}} \tag{4.2}$$

For simulation purposes the translational state covariance of both the satellite and the object are modeled as constant values. In a real system, it is expected that sensor measurements and error filtering will work to refine the translational state estimates, thereby decreasing the covariance values and allowing for longer robust trajectory segments. Whenever statistical information is estimated onboard the vehicle for error sources that impact motion planning, that information should be used in place of calibration data developed from Monte Carlo simulation. The real-time statistical data is pertinent to the specific motion planning problem encountered and may provide a

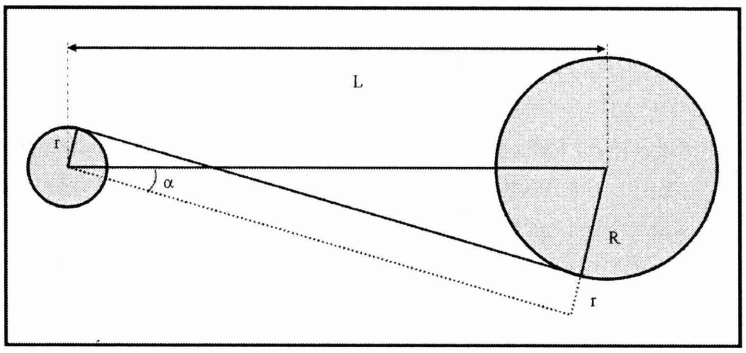

Figure 4-4: The geometric relationship used to calculated the pointing error induced by translational state propagation errors.

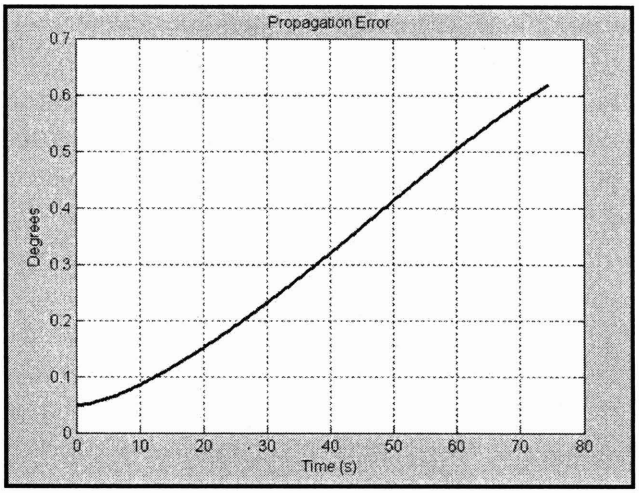

Figure 4-5: The growth of propagation errors in time.

lower upper-bound on the impact of the error than the analysis of challenging test cases.

Preplanner Summary

To review, the preplanner uses a combination of error source statistical data provided by the navigation system and error calibration data to arrive at a robust planning time horizon based on the maximum allowable constraint enlargement. The inputs to the preplanner implemented for this experiment include the current function call time, the time at which the next trajectory segment must begin, the error calibration data, the estimated translation state and estimation error covariance for the satellites and the object, and the maximum allowable degree of constraint enlargement. Using the error data and the maximum degree of constraint enlargement, the preplanner calculates the robust horizon cutoff time. The preplanner then propagates the satellites' and object's positions forward in time until the end of the next planning horizon, which for

the MPC inspired approach is the end of the engagement. Constraint boundaries are formulated based upon the propagated translational states and the maximum degree of constraint boundary enlargement. The next function call time is calculated by subtracting the expected computation time from the robust horizon cutoff time. The function call time is used to trigger the prelanner when it is time to begin computing the next segment of the plan. The initial planning conditions are simply the planned state of the vehicle at the end of the last robust trajectory segment calculated. For tracking problems like OTE, the end-plan condition can simply be set to a time far enough in the future to generate a plan that is unlikely to become entrapped by obstacles, but not so far that computation time becomes a factor. The preplanner outputs the constraint boundary definitions, the initial planning conditions, the end-plan conditions, and the next function call time.

4.2 Planner

The planning function is responsible for generating a plan based upon the initial plan conditions, the end-plan conditions, and the constraint boundary definitions supplied by the preplanner. Any open-loop planning algorithm capable of generating a timely feasible plan for the system within its environment will work. Two planning algorithms suitable for the satellite attitude guidance problem were introduced in Chapter 2; namely, Hsu's algorithm and Frazzoli's algorithm. The RSTP, a third and new algorithm that expands on the RRT is presented in this section. When an open-loop planner is incorporated into the guidance loop to solve receding horizon problems, the option exists to extend a trajectory segment plan beyond the robust horizon time. Issues concerning the use of these extended portions of the segment plan as a warm start for the following trajectory segment are presented in the first portion of this section, while the second portion of this section is reserved for the explanation of the RSTP algorithm.

4.2.1 Receding Horizon Warm Start

There are different approaches that to implementing an open-loop planner as a receding horizon planner. One approach is to set the planning horizon equal the robust horizon time and generate a new tree every time the planning function is called. A second and more favorable approach would be to extend the planning horizon beyond the robust horizon time, and use the extended portions of the tree as a warm start for developing the next robust trajectory segment. The warm start option works to speed up the convergence of the planner because a large component of the extended portion of the plan may be feasible in the development of the next robust trajectory segment.

The warm start option may be approached one of two ways. Either the solution of one trajectory segment may be saved, or the entire tree may be saved and used as part of the warm start process. Saving and using the only the solution provides the trunk of the new search tree, a single branch of the new search tree extending

from the root, or initial planning condition. When the planner is called to generate a trajectory segment, it first recalls the solution to the previous plan and checks to see which portions of the previous solution are feasible within the new planning times and updated constraint boundary definitions. The planner then uses the feasible portions of the old solution as the base of a new search tree aimed at generating the new segment. On the other hand, if the entire tree is saved, the new tree could start with several branches already in place. All branches that attach to the previous solution before the initial planning time of the new segment are deleted before the tree is checked against the new planning conditions and constraints. In this way, only the branches that stem from the previous solution after the new initial planning time are retained to be checked against the new constraints. After checking the old tree against the new constraints, only the feasible portions that attach to the initial planning conditions are saved. Refer to figure 4-6 for an illustration of a search tree that has been generated from an initial set of conditions to an end goal with a robust planning horizon. The dashed markings highlight the path selected for the robust horizon, while the portions of the tree that will be used as a warm start for the next planning segment are shown in solid black. The remainder of the search tree serves no purpose and is deleted. Once the warm start process is finished, the open-loop planning algorithm is initiated with either the initial planning conditions, or the beginnings of a search tree in place.

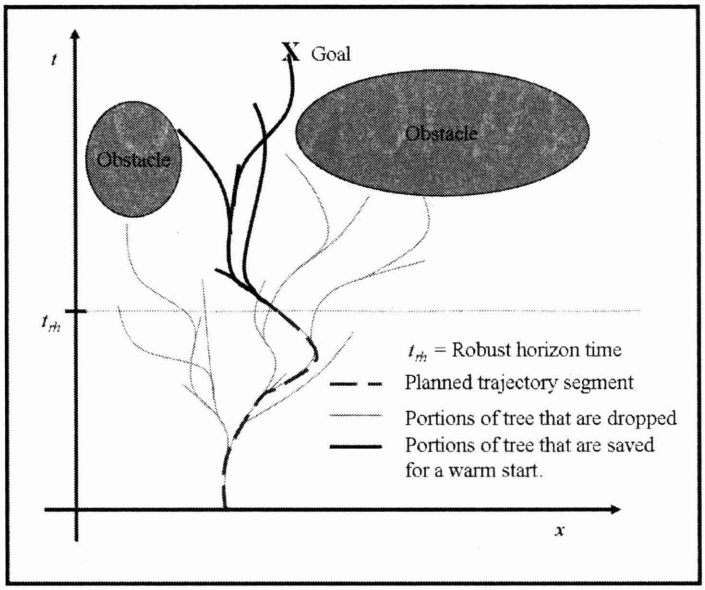

Figure 4-6: Tree used to provide the current trajectory segment and a warm start option for the following segment.

4.2.2 Randomized Spacetime Trajectory Planner

Recall that the success of the RRT algorithm was based upon its ability to rapidly and uniformly explore the free-space of a problem with static obstacles. The rapid,

uniform exploration of the configuration space was accomplished by choosing random configurations to draw the exploration of a search tree toward unexplored regions of the configuration space. It was acknowledged in [22] that motion planning problems in dynamic environments extends the problem an additional dimension to include time, so the motion planning problem is approached by searching the configuration-$\times$-time space. The natural extension of the RRT to dynamic environments is to use random configuration-$\times$-time points to draw the exploration of a search tree into a uniform coverage of the configuration-$\times$-time space. The difficulty with extending the RRT in this way lies in the definition of a metric used to measure distance or control effort in configuration-$\times$-time space. In order to circumvent this difficulty, Hsu's algorithm explores the configuration-$\times$-time space by choosing a node of the tree at random according to a sampling strategy that divides the configuration-$\times$-time space into bins. From the selected node, Hsu's algorithm applies a random control input to extend. Using a different strategy, Frazzoli's algorithm attempts to explore the configuration-$\times$-time space by choosing a random configuration from a uniform distribution and attempting to connect the random configuration to every node in the tree until a connection is successfully made. Both Frazzoli's and Hsu's algorithms work to extend the RRT algorithm to problems with dynamics environments; however, neither algorithm uses random points in configuration-$\times$-time space to guarantee a search tree that uniformly explores the space of the problem.

Both Hsu's algorithm and Frazzoli's algorithm claim probabilistic completeness under a reasonable set of conditions; although, a notable condition exists on the probabilistic completeness of Frazzoli's algorithm. Frazzoli's algorithm is probabilistically complete if the original problem is π_r reachable, meaning that if the goal can be obtained through a series of rest to rest maneuvers constructed using the control policy π_r, then the algorithm will find a feasible path. However, the rest to rest maneuvers generated by π_r may only represent a subset of all the maneuvers the vehicle is capable of executing. Therefore in some problems the vehicle will have the capability of obtaining the goal; however, the planning algorithm will not successfully produce a feasible path. In order for an algorithm to be probabilistically complete without the π_r reachability condition, the basis of maneuvers selected for growing the tree must span the entire maneuver space.

Therefore, an ideal extension of the RRT algorithm to problems with moving obstacles would uniformly explore the configuration-$\times$-time space through the selection of random configuration-$\times$-time points, while employing maneuvers from a basis that spans the entire maneuver space. Two major obstacles stand in the way of implementing this ideal extension of the RRT algorithm. The first obstacle is determining a metric that can be used to select nodes in the tree for expansion toward randomly selected configuration-$\times$-time points. The second obstacle is developing a control strategy to connect a node in the tree, to a given configuration-$\times$-time point, while drawing from a maneuver basis that spans the maneuver space. Neither obstacle is easily overcome for general application to any given system. However, the following algorithm is presented with the hope that the conceptual aspects may be generally applied, while the specific details are developed only for fully actuated, fully separable systems, where the dynamics in each dimension can be described independently from

the dynamics in the other dimensions.

Analogy to Spacetime Physics

Before going into the details of the RSTP, the notion of the spacetime diagram must be introduced. The spacetime diagram is often used to illustrate the concepts of Lorentz geometry and the fundamental precepts of Special Relativity [48]. A typical spacetime diagram displays a spatial dimension along the abscissa and the temporal dimension along the ordinate. There are several important differences between a diagram of Euclidean space and a spacetime diagram so in order to maintain a cognitive separation of parallel concepts some basic terms from relativity theory are defined. In Euclidian geometry a *point* is denoted by a set of coordinates; on a spacetime diagram, a set of coordinates denotes an *event.* Secondly, the invariant interval between two points in Euclidean space is called the *distance* between the points, where the invariant interval between two events is referred to as the *spacetime interval.* Equation 4.3 compares the equations defining distance, d and the spacetime interval, s, where c is used to represent the speed of light. Third, a line denoting the location of an object at different times in Euclidean space is called a *path*, while the same line plotted on a spacetime diagram is referred to as a *worldline.* The cumulative sum of the distances between consecutive points on a line is known as the *length* of the line, while on a spacetime diagram, the cumulative sum of the intervals between events on a world line is referred to as the *proper time* of the world line [48]. Figure 4-7 compares the concepts just introduced on a Euclidian graph and a spacetime diagram.

$$\begin{aligned} d^2 &= x^2 + y^2 + z^2 \\ s^2 &= c^2t^2 - (x^2 + y^2 + z^2) \end{aligned} \tag{4.3}$$

Notice how the distance between two different points will always be positive, while the squared spacetime interval between two different events could be positive, negative, or zero. Recall that the spacetime interval was described as an invariant interval, meaning that spacetime interval between two events will be the same regardless of the inertial reference frame chosen to describe the events. The spacetime interval can be broken down into three classifications based upon the sign of the squared spacetime interval. If the squared interval is positive, it is known as *timelike*, if the squared interval is zero it is referred to as *lightlike*, and if the squared interval is negative then it is called *spacelike.* If a spacetime interval between two events is lightlike, it can be seen that only something moving with the speed of light is able to connect the two events. It follows then that two events that are spacelike are completely independent of one another in that no cause and effect relationship can exist between the two events because no information or physical object can travel faster than the speed of light. Therefore, two events can share a cause and effect relationship only if the spacetime interval between them is timelike [45].

Starting from the different classifications of the spacetime interval, conditions for defining whether or not an event is reachable for a vehicle with a particular position

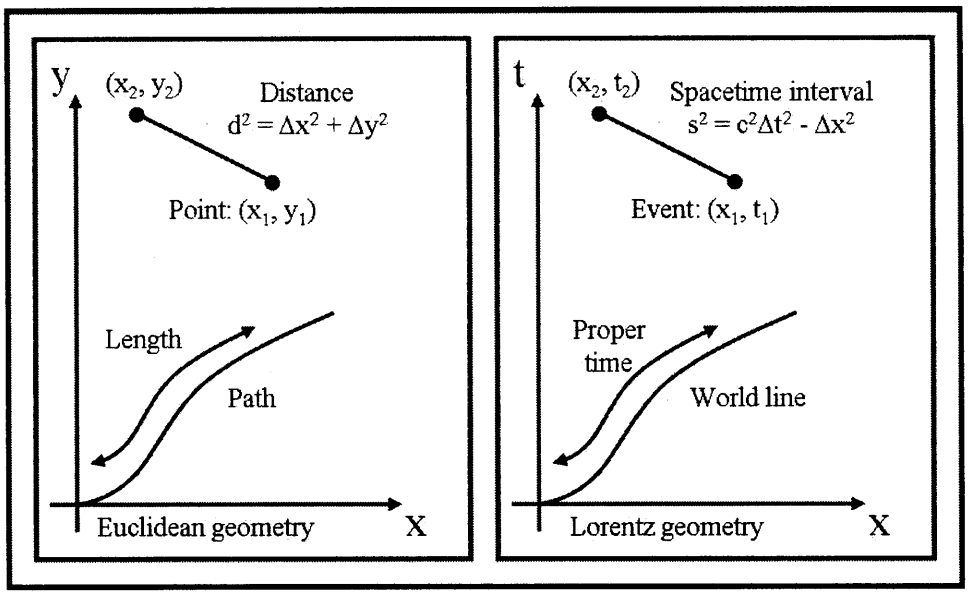

Figure 4-7: The parallels of Euclidean geometry and Lorentz geometry.

and velocity (state) at a particular time can be derived within the spacetime context. In order for a vehicle to move from its current configuration, or initial event, to another specified configuration and time, or final event, the two events must share a cause and effect relationship, which is the connection of the events via the world line of the vehicle. This notion suggests that a necessary condition for establishing the reachability of a randomly chosen event from the state of a vehicle at a specified time is that the random event and the event defined by the vehicle's state share a timelike interval. The sufficient condition required to assert that a randomly chosen event is reachable from the vehicle's state at a particular time, is that the vehicle must be able, through a series of allowable control actions, to produce a world line that connects the event defined by the vehicle's initial state and time to the randomly chosen event.

Reachable Events

To show how the reachable set of events can be defined for a vehicle with a particular position and velocity, an example is provided for a particle confined to movement in one dimension with velocity and acceleration constraints. Consider a particle, p, that is confined to travel along the x axis with an initial position at the origin and an initial velocity equal to 0. Assume also that the particle is confined to accelerations between $-a$ and $+a$, and velocities between $-v$ and $+v$. The border of reachable events along the positive x axis is found by employing the maximum acceleration until the maximum velocity is achieved. A similar border of reachable events is established along the negative x axis by choosing the most negative acceleration and velocity that the vehicle is capable of executing. Let the world lines established by executing the vehicle's maximum traveling capability in either direction be known

as maximum capability world lines. A randomly chosen event is reachable by the particle if the temporal component of the random event is larger than the temporal component of the event on the maximum capability world line that corresponds to the spatial component of the random event. On the spacetime diagram, the reachable set corresponds to all events on or above the maximum capability world lines (See Figure 4-8). The same definition of the reachable space applies to the particle if it has an initial velocity. However, notice if the initial velocity is positive, then the reachable set of events with positive spatial components will be larger than the reachable set with negative spatial components and vice versa (See Figure 4-8).

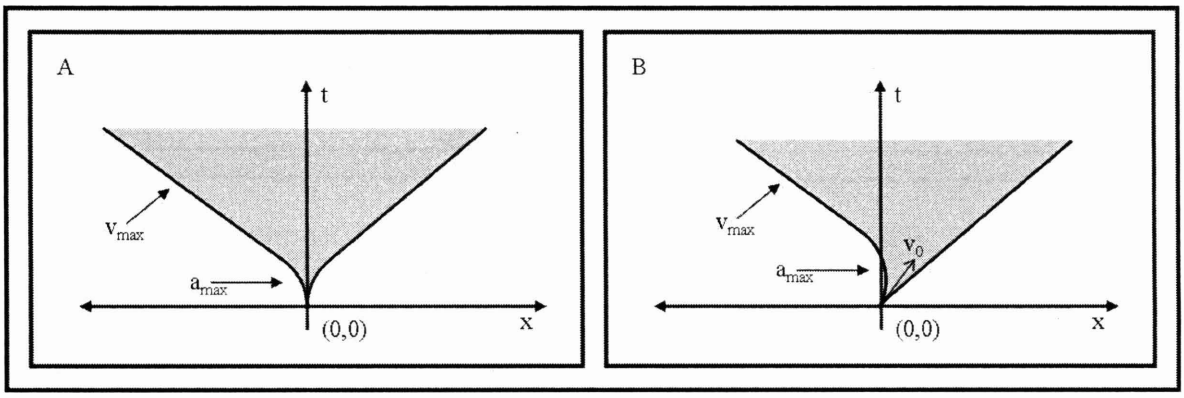

Figure 4-8: The reachable space (shaded) for a static vehicle at the origin (A) and for a vehicle with an initial velocity (B).

In order to extend the RRT algorithm to problems with moving obstacles, the RSTP uses randomly selected events to draw a search tree into unexplored, reachable regions of spacetime. In using randomly selected events to expand a search tree it is important to be able to define the reachable set of events from a particular vehicle state and time because unlike the RRT or Frazzoli's algorithm where in the obstacle free case random configurations can be achieved from any node, the RSTP chooses random events that may not be reachable from any node even without obstacles to block the attempt to connect events. Additionally, the concept of reachability suggests that when selecting random events to expand a search tree, the random event should be reachable (under the obstacle free assumption) by at least one node in the tree and that only the nodes that contain the random event within their reachable space should be considered for expansion.

Metric and Expansion of Tree

A set of baseline maneuvers is chosen for two purposes; one, to serve as a metric for selecting a node to expand from the tree, and two, to serve as a control strategy that will connect a randomly chosen event to the search tree. There are two requirements placed upon the set of baseline maneuvers. The first requirement is that the entire maneuver space must be spanned by the *stacking* of baseline maneuvers. When an

initial baseline maneuver is selected, a second baseline maneuver is *stacked* upon the first when the second maneuver is performed any time after the process of executing the first maneuver is initiated. Successive maneuvers can be stacked upon one another such that the system begins to execute the first part of the first baseline maneuver, then switches into the first part of the second baseline maneuver, and before the second baseline maneuver is completed, the system once again switches into the third baseline maneuver. The second requirement placed upon the set of baseline maneuvers is that given the state of the vehicle and a reachable, randomly selected event, a finite series of baseline maneuvers can be chosen to produce an executable world line that connects the event defined by the initial vehicle state to the randomly selected event.

Node Selection

In addition to acting as a control strategy to connect a randomly selected event to the tree, the baseline maneuvers are used as a metric for expanding the tree. Before the metric is described, the concept of a resultant world line must be defined. When a baseline maneuver is applied to a given node the world line extending from the node will be referred to as the *resultant* world line. Now, to choose a node in the tree for expansion, a random baseline maneuver (or maneuvers) is selected along with a random event. The selection of a node is accomplished by comparing the baseline maneuver required to connect the node to the random event with the randomly selected baseline maneuver. The node where this comparison results in the smallest difference is chosen for expansion, and the baseline maneuver that will successfully connect the node to the random event is selected as the means of expansion. In this way, the metric is based upon the vehicle's control effort and the search tree uniformly explores the reachable spacetime while also encouraging a uniform exploration of the maneuver space.

RSTP Algorithm Description

The structure of the RSTP is very similar to the RRT algorithm. Like the RRT algorithm, RSTP is built upon 6 chief components: the configuration space, the state space, the boundary conditions, a set of baseline maneuvers, an incremental simulator, and a collision detector. The configuration space captures the essence of the position of the vehicle or system. The state space contains the configuration space and the first derivatives of the configuration parameters. The boundary conditions specify the vehicle's initial state, the end-plan conditions, and the edges of the spacetime box to be explored. The set of baseline maneuvers can be stacked to span the maneuver space of the system or vehicle and an identifiable set of baseline maneuvers can be used to connect the vehicle from any given state to any reachable random event. An incremental simulator propagates the vehicle state using the control actions selected. Finally, the collision detector processes the simulated vehicle state and returns where collisions with obstacles occur.

Algorithm Flow

Initially, an attempt is made to connect the initial state to an event included within the end-plan conditions. If the attempt is successful, the algorithm returns the successful trajectory. If the attempt is not successful, then a random event is selected. The random event is screened to ensure both that it is reachable (in the obstacle free sense) by the root of the tree (the initial state), and that the end-conditions are reachable from the random event. If the random event is not reachable by the root, or cannot reach the end-plan conditions then it is discarded, and a new random event selected. Next a random baseline maneuver or set of baseline maneuvers is selected to use as a metric for determining which node to expand. At each node that can reach the random event, the comparison is made between the world line resulting from application of the random baseline maneuver and the baseline maneuver required to connect the node to the random event. The node where the comparison results in the smallest difference is chosen. The control actions required to implement the baseline maneuver that connects the selected node to the random event is sent to the incremental simulator. The incremental simulator propagates the system state while the collision detector verifies that each state produced by the simulator is collision free. The propagation/collision checking process continues until the event is reached or a collision occurs. For each successful increment simulated, a new node is placed. A safety time can be introduced at this point where a node is not allowed to be placed a certain time before a collision occurs. At this point a new random event is chosen and the process used to expand the tree starts all over.

Parallels to RRT and Data Storage

The same techniques used to improve convergence for the RRT also apply for this algorithm, such as biasing the search toward the goal, and growing a bidirectional tree. The previous paragraph describes the algorithm in light of the CONNECT approach used in the RRT, where a branch of the tree is extended until the random event is obtained or a collision occurs. The EXTEND approach is just as easily applied, where branch of the tree is extended one incremental step for each random event selected. The notions of edges and vertices introduced for data storage within the RRT also apply to the RSTP. The state of the vehicle, the time, and the address of the parent node are stored at each node or vertex, while the node address, control actions between nodes, and any other feedforward parameters are stored at each edge. Figure 4-9 depicts a sample tree generated by the RSTP formed in two spatial dimensions and time, while Table 4.1 provides the pseudocode for the algorithm.

Table 4.1: RSTP Pseudocode

BUILD_RSTP_TREE (x_{init})

1. $\mathcal{T} = x_{init}$
2. **for** $n = 1$ **to** N **do**
3. **Choose** random event and baseline maneuver
4. **if** random event is reachable from x_{init}
5. Sort nodes according to maneuver metric
6. **else**
7. Return to select new random event
8. **Select** node where maneuver to event most closely resembles random baseline maneuver.
9. BUILDBRANCH ($\mathcal{T}, node, event$)
10. **Return** $\mathcal{T}$

BUILDBRANCH($\mathcal{T}, node, event$)

1. m = CalculateManeuver (node, event)
2. **repeat**
3. $x(t)$ = NewState(m, node)
4. [T, F] = ConstraintViolation($x(t)$)
5. **until** ConstraintViolation = T
6. $t = t_{node}$
7. **repeat**
8. $t = t + t_{node\_interval}$
9. $x(t)$ = new node
10. **until** $t = t_{violation} - t_{safety\_factor}$
11. **Return** $\mathcal{T}$

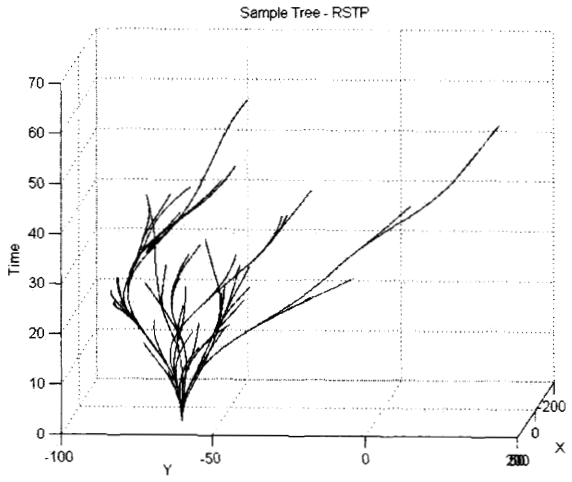

Figure 4-9: Example of a search tree generated by the RSTP

RSTP Algorithm Application to 3-D Attitude Guidance

The specific application of the RSTP to the OTE is significantly simplified by the fact that the dynamics of each dimension of satellite attitude guidance problem can be considered separately since the inner loop controller supplies the commands to negate the gyroscopic forces that couple rotational maneuvers. Furthermore, the satellite is fully actuated, so control actions can be applied to each of the three rotational dimensions independently. Therefore, the application of the algorithm can be examined for one dimension and then applied to the other dimensions.

State, Initial, and End-Plan Conditions The configuration space of the OTE attitude guidance problem consists of the parameters required to define the attitude of the satellite. Euler angles, defined by X-Y-Z rotations, are used so the problem can be approached one dimension at a time. The state space includes both Euler angles and the angular velocity of the satellite with respect to an inertial frame, formulated in the body frame. The initial conditions consist of the initial satellite rotational state and time. The end-plan conditions are characterized by a time which must fall between the robust horizon planning times and the time at which the engagement terminates. Because OTE is a tracking problem, the planner is only required to produce a solution that remains feasible as long as the satellite can continue tracking the object, which means there is not a particular event or region of events which constitutes the end-plan condition.

Problem Boundaries The boundaries of the configuration space are for roll, $0 \leq \phi \leq 2\pi$, for pitch, $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$, and for yaw, $0 \leq \psi \leq 2\pi$. The lower boundary placed upon time is the time associate with the satellite's initial state. The upper boundary placed upon time is selected to be a linear function of the maximum time covered by the tree. That is to say that $t_0 \leq t \leq t_0 + t_c + r(t_{\mathcal{T}(max)})$ where t_c is a positive constant, r is a receding horizon scaling factor, and $t_{\mathcal{T}(max)}$ is the maximum time of all the nodes in the tree. Note that when $t_{\mathcal{T}(max)}$ meets or surpasses the end-plan condition, the algorithm returns with the trajectory that spans the engagement time. The upper boundary on time was chosen this way in order to continue to draw the search tree up in time. The constant t_c exists for the initial growth of the search tree when $t_{\mathcal{T}(max)}$ is equal to t_0. The scaling constant, r, effects the balance between promoting exploration of the configuration space (outward) and promoting exploration of the time space (upward).

Baseline Maneuvers The baseline set of maneuvers and the method of selecting a node to expand is first approached in one dimension of the problem, and then applied to the other dimensions. A baseline maneuver for one dimension of this problem is characterized by a period of acceleration, constrained by the capability of the satellite, followed by a period of coasting at a constant velocity. The set of baseline maneuvers is not a countable set, though a single maneuver can move the system from a given state to a reachable, random event. Figure 4-10 shows a velocity profile of two baseline maneuvers and the corresponding spacetime diagram.

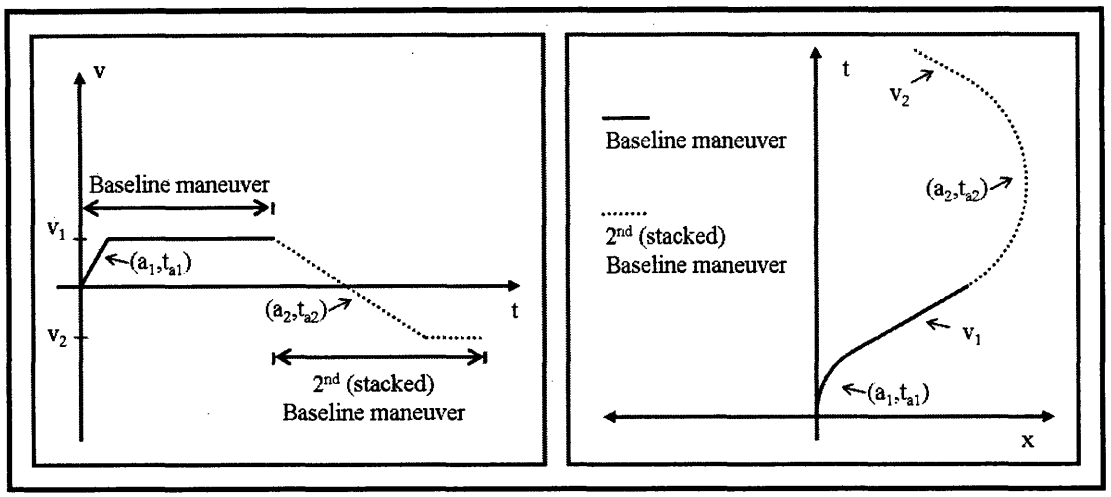

Figure 4-10: Two baseline maneuvers (one stacked on the other) depicted both in a velocity profile (left) and in a spacetime diagram (right).

When using maneuvers to move from the vehicle state to a randomly selected event within a rotational environment, one must be conscience of angular distances in regards to the limits placed on the values of the Euler angles. For example, $\phi_0 = \frac{\pi}{6}$ is closer to $\phi_f = \frac{11\pi}{6}$ by rotating through $\phi = 0$ then by rotating through $\phi = \pi$. Therefore, to produce consistent results, the following convention is set: whenever a control input is chosen to move from an initial state to a random event, the sign of the control input is selected based upon the rotation resulting in the shortest angular distance traveled.

Node Selection and Tree Expansion To expand the tree, a random event is selected that lies within the obstacle-free reachable set of the root of the tree. At a minimum the random event must lie in the obstacle-free reachable set of the tree root because the obstacle-free reachable set of the root contains the obstacle-free reachable set of every other node in the tree. Table 4.2.2 presents the logic that is used to screen the randomly selected event to ensure that it is reachable by the root. The variables $\dot{\phi}_{max}$ and $\ddot{\phi}_{max}$ denote the maximum allowable angular velocity and acceleration, while the variables $\dot{\phi}_{ext}$ and $\ddot{\phi}_{ext}$ denote the extreme limit of the vehicle's capability in the direction the control action is applied. The difference between $\dot{\phi}_{max}$ and $\dot{\phi}_{ext}$ is that $\dot{\phi}_{ext}$ can take on a negative value. The random event is denoted by (ϕ_r, t_r) and the vehicle state at the root of the tree is given by $(\phi_0, \dot{\phi}_0, t_0)$. Note that Table 4.2.2 is only written for one dimension. To apply the screening test to the entire problem, then the random event must be reachable for every dimension.

When a random event is selected that is reachable from the root, the same screening process is applied to the remaining nodes in the tree, and only the nodes that can reach the new random event are considered for expanding the tree. Once the subset of nodes that can reach the random event is determined, a baseline maneuver is selected to serve as a metric for selecting the next node to expand the tree.

Table 4.2: Screening Logic

If $|\phi_r - \phi_0| > \pi$
  If $\phi_0 > \phi_r$
    $\phi_r = \phi_r + 2\pi$
  Else
    $\phi_r = \phi_r - 2\pi$
End
If $\phi_r - \phi_0 \leq 0$
  $\dot{\phi}_{ext} = -\dot{\phi}_{max}$
  $\ddot{\phi}_{ext} = -\ddot{\phi}_{max}$
End

If $t_r - t_0 < \frac{\dot{\phi}_{ext} - \dot{\phi}_0}{\ddot{\phi}_{ext}}$

  If $\frac{\dot{\phi}_0(t_r - t_0) + \frac{1}{2}\ddot{\phi}_{ext}(t_r - t_0)^2}{\phi_r - \phi_0} > 1$

    ϕ_r = reachable
  Else
    ϕ_r = unreachable

Else $t_r - t_0 \geq \frac{\dot{\phi}_{ext} - \dot{\phi}_0}{\ddot{\phi}_{ext}}$

  $t_{a_{max}} = \frac{\dot{\phi}_{ext} - \dot{\phi}_0}{\ddot{\phi}_{ext}}$

  If $\frac{\dot{\phi}_0(t_r - t_0) - \frac{1}{2}\ddot{\phi}_{ext}t_{a_{max}}^2 + \ddot{\phi}_{ext}t_{a_{max}}(t_r - t_0)}{\phi_r - \phi_0} > 1$

    ϕ_r = reachable
  Else
    ϕ_r = unreachable
End

The baseline maneuver is specified by choosing a random value of acceleration and a random time to implement the acceleration. The acceleration value for the random maneuver, a_{rm}, is chosen according a uniform distribution of all possible accelerations. The random time value, t_{rm}, is chosen according to a uniform distribution from zero to twice the time required to accelerate from a rest to the maximum acceleration. In this way, when a median positive (negative) acceleration is paired with a median value of time, the resulting change in angular velocity is the median of the positive (negative) angular velocity range.

To sort the nodes, the acceleration required to attain the random event from each

node is calculated based upon the randomly selected maneuver time, (t_{rm}). The calculation used to determine the acceleration values for each node, n, is given by Equation 4.4.

$$a_n = \frac{\phi_r - \phi_n - \dot{\phi}_n(t_r - t_n)}{-\frac{1}{2}t_{rm}^2 + t_{rm}(t_r - t_n)} \tag{4.4}$$

For each node, the acceleration required to obtain the random event, a_n is compared to the randomly selected acceleration, a_{rm}, using Equation 4.5. The node with the smallest difference in acceleration values, Δa_n, is then selected for expanding the tree. To adapt the node selection to account for every dimension, Equation 4.6 is employed, where K represents the total number of dimensions, which in this case is three.

$$\Delta a_n = |a_{rm} - a_n| \tag{4.5}$$

$$\Delta a_{n_{total}} = \sum_{k=1}^{K} \Delta a_{n_k} \tag{4.6}$$

The acceleration values used to generate feedforward commands, a_{ff} are initially set to the selected random accelerations a_{rm}. The magnitude of a_{ff} for each dimension are then screened against the minimum magnitude of acceleration required to reach the random event. The calculation used to determine the minimum magnitude of acceleration, $|a|_{min}$, required is provided in Equation 4.7. The value of a_{ff} is adjusted to equal the minimum magnitude of acceleration for any case where $|a_{ff}|$ is less than $|a|_{min}$. Using the (adjusted) feedforward acceleration values, the corresponding acceleration times that will result in a maneuver which will move the system from the node to the randomly selected event are calculated. The calculation used to calculated the acceleration time, t_a, is given in Equation 4.8. Next, a time history of feedforward torques is calculated from the feedforward acceleration values and their corresponding acceleration times. The torque, $T_{ff}(k)$, calculated at each time step is formed using Equation 4.9.

$$|a|_{min} = \frac{(\dot{\phi}_{ext} - \dot{\phi}_n)^2}{|2((\dot{\phi}_{ext})(t_r - t_n) - (\phi_r - \phi_n))|} \tag{4.7}$$

$$t_a = t_r - t_n - \sqrt{(t_r - t_n)^2 + 2\frac{\dot{\phi}_n}{\ddot{\phi}}(t_r - t_n) - 2\frac{\phi_r - \phi_n}{\ddot{\phi}}} \tag{4.8}$$

$$T_{ff}(k) = \hat{\mathbf{J}} \begin{bmatrix} \ddot{\phi}(k) \\ \ddot{\theta}(k) \\ \ddot{\psi}(k) \end{bmatrix} \tag{4.9}$$

Collision Checking and Data Storage The time history of feedforward torques that will bring the satellite through the maneuver is then sent to an incremental

simulator which constructs a time history of the satellite states. The satellite state values are checked against the constraints using a collision checker. The incremental simulator and collision checker process the maneuver until a collision occurs or the maneuver is complete. If no collision occurs, then new nodes are created at incremental times between the node and the event. On the other hand, if a collision occurs, new nodes are created at incremental times from the node to a safe time before the collision occurs. For this experiment, nodes were generated every 2 seconds of collision free maneuvering outside of a safety time of 5 seconds before a collision occurs. Feedforward torque, angular velocity, and quaternion values are stored along each edge, while the rotational state of the satellite, the time, and the parent node address are stored at each node.

Biasing Toward Goal To improve convergence, the sampling of random events was biased towards the goal. Because the goal is a collision free trajectory that extends the length of the engagement time, biasing toward the goal means extending the tree toward the end of the engagement time. This amounts to creating vertical branches on the tree depicted on a spacetime diagram. To accomplish the generation of vertical branches, every node with an angular velocity above a certain threshold is flagged as a *moving* node. When it comes time to expand the tree, a random descision variable is added to choose to either select a new random event, or bias the tree expansion toward the goal. The decision variable was designed to select random events 80% of the time, and bias the growth of the tree 20% of the time. When the decision is made to bias the growth of the tree, a moving node is selected at random from all moving nodes within the tree and the control input required to bring the vehicle to a rest is applied to the node. If no moving nodes exist, a random event is selected just as though the decision were made to expand the tree with a random event.

4.2.3 Summary

The preplanner and the planner work together in the outer, planning loop to provide timely, robust guidance solutions to maneuver a vehicle through a dynamic environment. The outer, planning loop runs at a much lower rate than then inner, execution loop by providing trajectory segments that are typically tens to hundreds of seconds in duration. The preplanner and the planner have been described both in general terms to promote application to a broad range of problems, and in the specific context of satellite attitude guidance to provide a concrete example of the implementation of the RRHC and the RSTP.

Chapter 5

Results

This chapter discusses the successful execution and limitations of both of the RRHC and the RSTP applied to satellite attitude guidance. Simulation results for the for the first engagement scenario are provided with a detailed explanation of each figure. Engagement scenarios two and three are then presented in the same manner as the first though with less explanation. A brief comparison is made with Frazzoli's algorithm that illustrates an advantage of the RSTP. Finally, the results of an open-loop solution applied to the first engagement scenario are presented to compare with the results of the RRHC.

5.1 Engagement Scenario 1

The engagement scenario parameters are generated with the aid of the Satellite Tool Kit (STK) and are provided in 3.4. Figure 5-2 illustrates the two OTE satellites and the object that is tracked. Dashed lines extending from each satellite and the object illustrate the paths which they take for the 360 second engagement. The navigation system uses a star camera update rate of 0.2 Hz and the estimated moment of inertia values are perturbed by 1% of their true values (refer to Equation 3.65). The reaction wheels are capable of producing a maximum angular acceleration of $0.15 deg/s^2$ and a maximum angular velocity of $3 deg/s$. Figure 5-1 depicts the hardware simulated within the experiment. The other system parameters provided in Chapter 3 are used to produce the following results.

Figures 5-3 and 5-4 illustrate the free-space of the motion planning problem presented by this engagement scenario. The X and Y axes of Figure 5-3 denote the Euler angles of pitch and yaw of the satellite. The Z axis represents time in seconds and the tube represents the constraint placed upon the primary camera to point at the object. The bore-sight of the primary camera must remain within the tube throughout the engagement. Because the tube is very narrow compared to the rest of the field, the problem is considered a highly constrained problem. Figure 5-4 illustrates the free-space in the roll dimension. To generate the depictions of the free-space in the roll dimension, it was assumed that the primary camera always points directly at the object. An arbitrary initial roll position was chosen to generate the graph. The

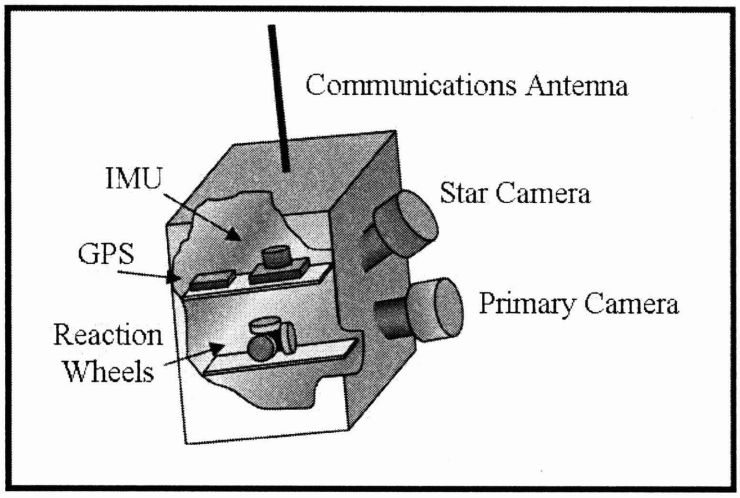

Figure 5-1: The sensors and hardware of each OTE satellite.

ordinate of the graph shows the roll angle from 0 to 360 degrees, and the abscissa denotes the time of the engagement. The shaded regions illustrate the roll angles that will result in a violation of the constraints; which are essentially the intersection of moving exclusion cones with a roll plane that is also moving. This graph is not a perfect illustration of the free-space in the roll dimension, because the primary camera is allowed to drift away from the line-of-sight vector to the object by a small amount and the free-space boundaries in the roll dimension will change when the primary camera drifts from pointing directly at the object. However, because the primary camera is confined to a narrow region around the line-of-sight vector to the object, the free-space displayed is a reasonable approximation for illustrative purposes.

The search trees used in the generation of the robust trajectory segments are saved and combined to produce Figures 5-5, 5-6, and 5-7 which illustrate the search tree's growth in the roll, pitch and yaw dimensions throughout the engagement. Some trajectory planning segments proved more challenging than others resulting in some portions of the tree which are heavily populated by branches and other portions which are relatively sparse.

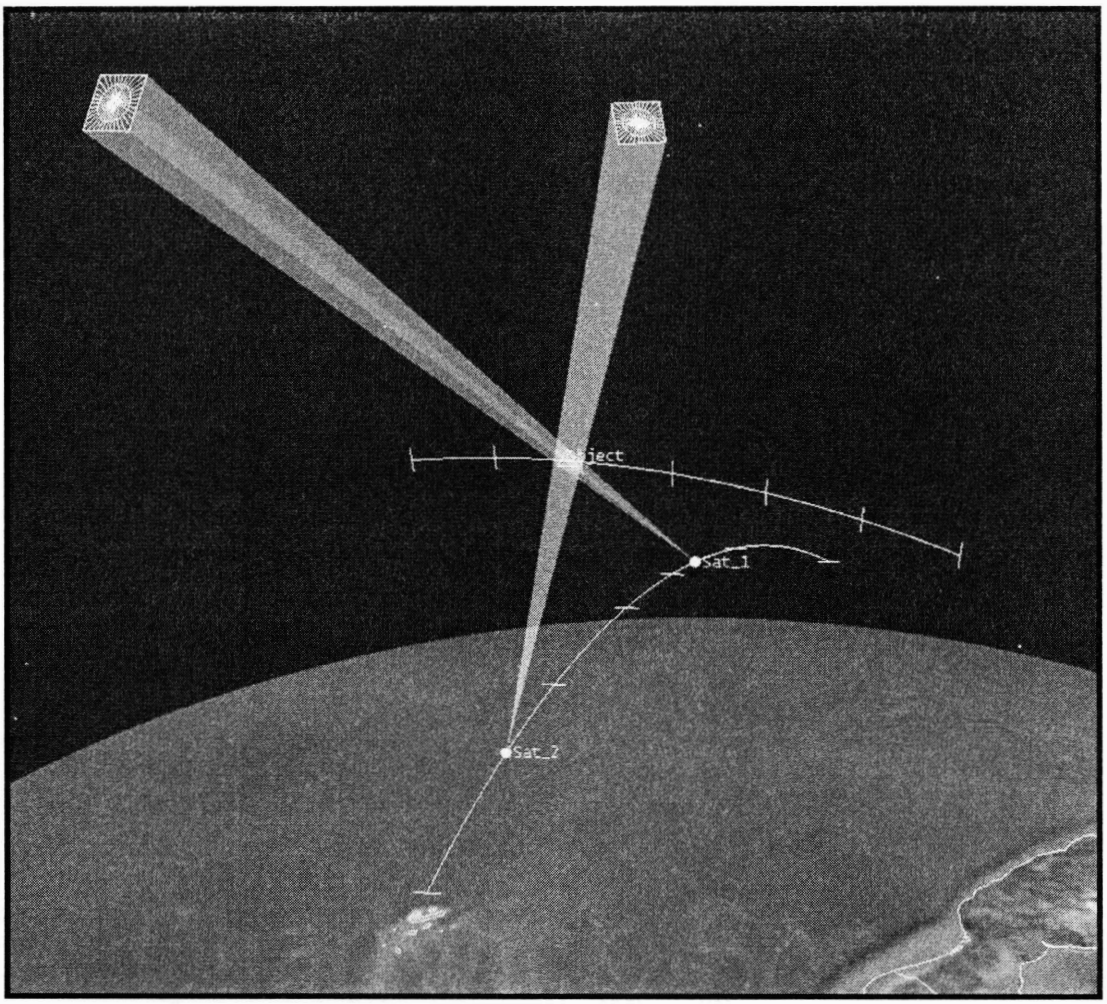

Figure 5-2: Engagement Scenario 1 illustration of orbits.

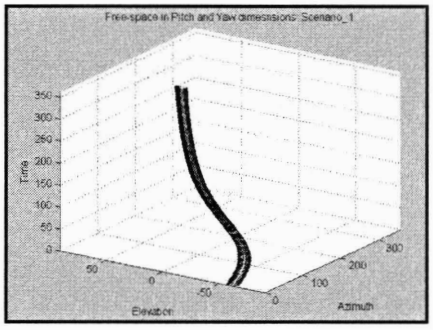

Figure 5-3: Free-space of the pitch and yaw dimensions of Scenario 1.

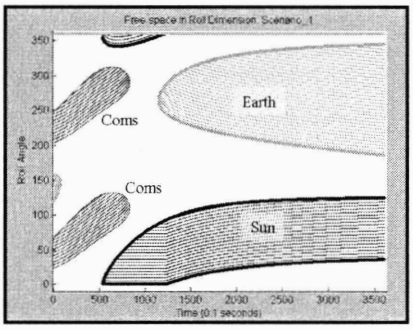

Figure 5-4: Free-space of the roll dimension of Scenario 1.

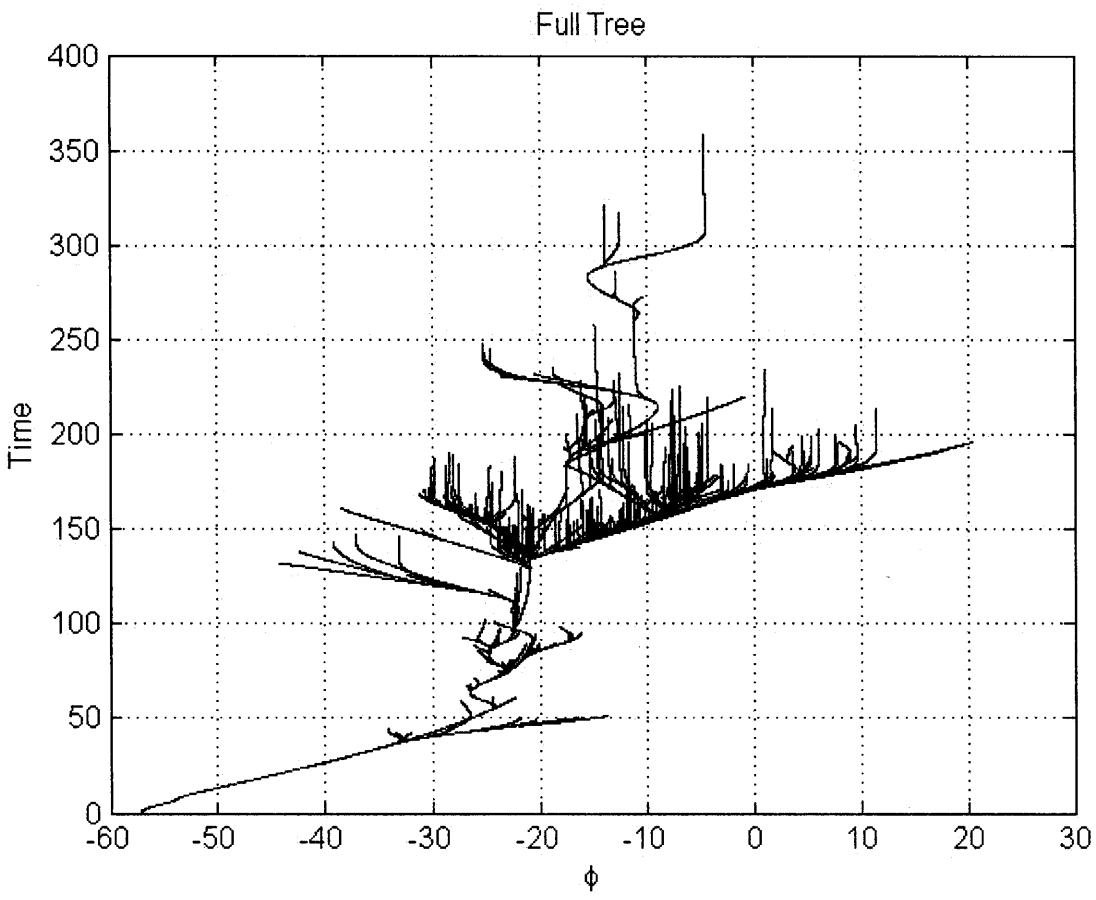

Figure 5-5: Search tree produced in Engagement Scenario 1 viewed in the roll dimension.

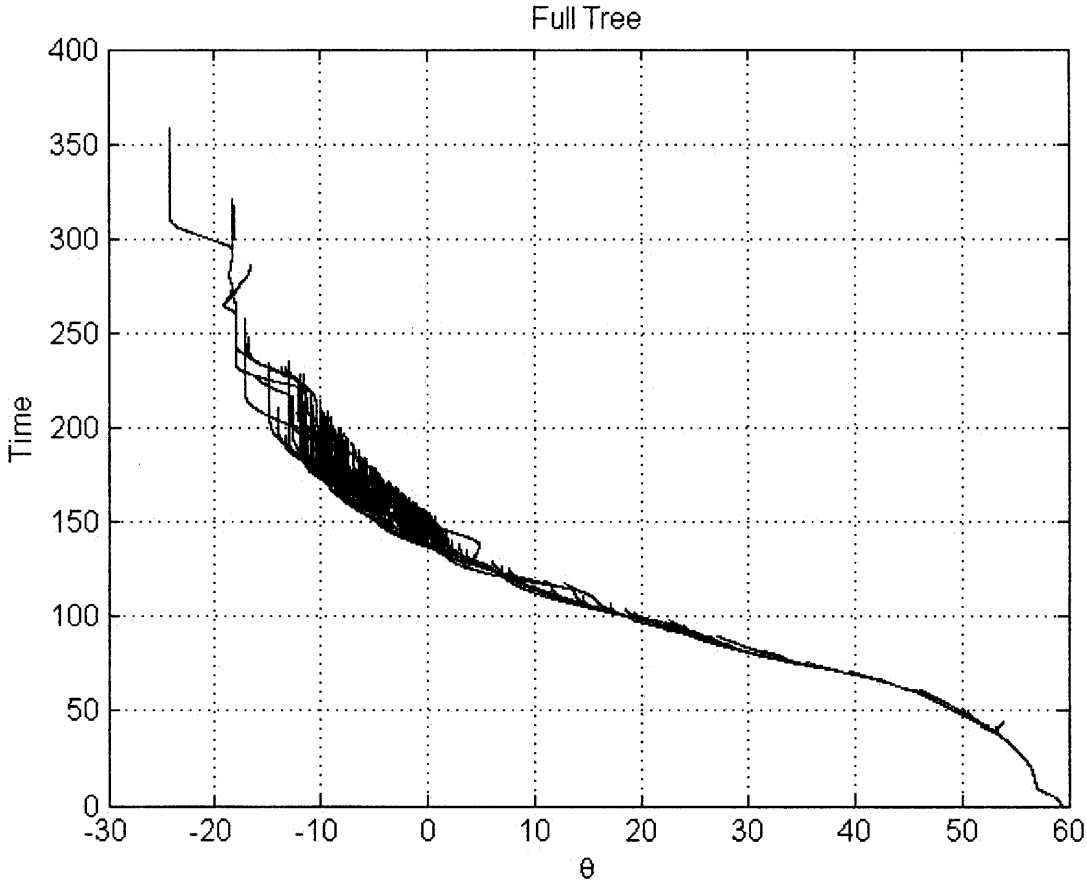

Figure 5-6: Search tree produced in Engagement Scenario 1 viewed in the pitch dimension.

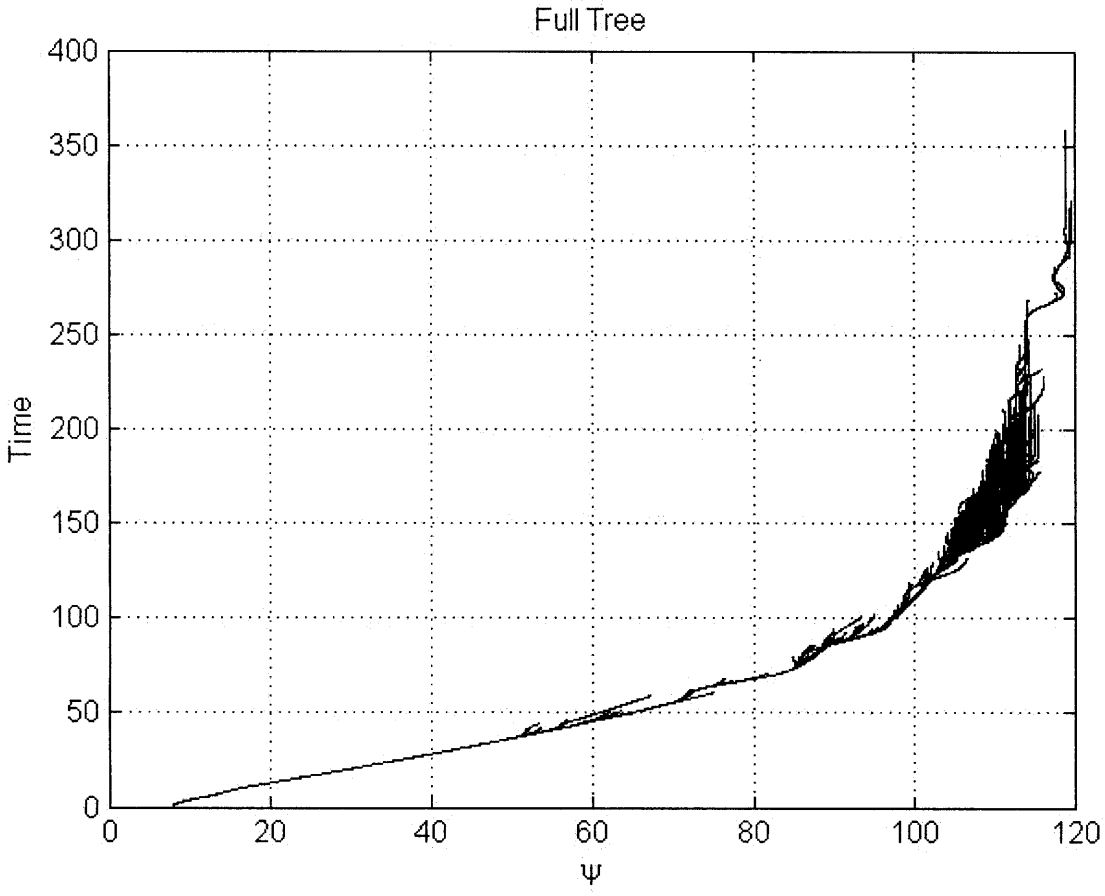

Figure 5-7: Search tree produced in Engagement Scenario 1 viewed in the yaw dimension.

Figures 5-8, and 5-9, are used to verify that the constraints of the problem were satisfied throughout the engagement. Figure 5-8, illustrates the exclusion cone constraints of the star camera and communications antenna. On the star camera constraint plots, the ordinate provides the angle between the star camera bore-sight and the pointing vectors indicating the relative location of the Sun, Earth and Moon. Similarly, the ordinate of the communications antenna constraint plot depicts the angle between the antenna and the pointing vector indicating the relative position of the second satellite. For the primary camera constraint plot (Figure 5-9), the angle between the primary camera bore-sight and the line-of-sight vector to the object is plotted over time. The dashed horizontal lines indicate the location of the artificial constraint boundaries used by the planner. The thin solid line indicates the angle expected by the planner while the heavy dotted line is derived from the true motion of the satellite. The difference between the thin solid line and the heavy dotted line is the result of the errors in the system. If the RRHC is successful then the anticipated motion of the satellite will not cross an artificial constraint boundary and although the true motion of the satellite may violate an artificial constraint boundary it will never cross the true constraint boundary.

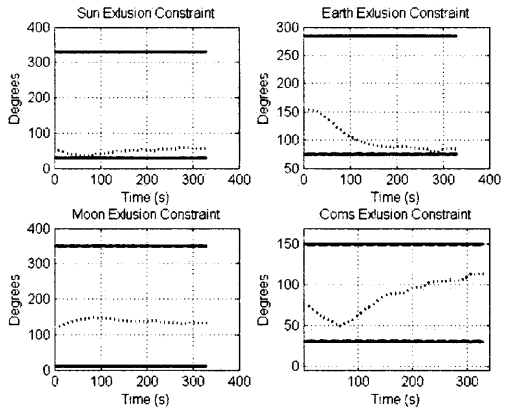

Figure 5-8: Communications and star camera constraint satisfaction for Scenario 1.

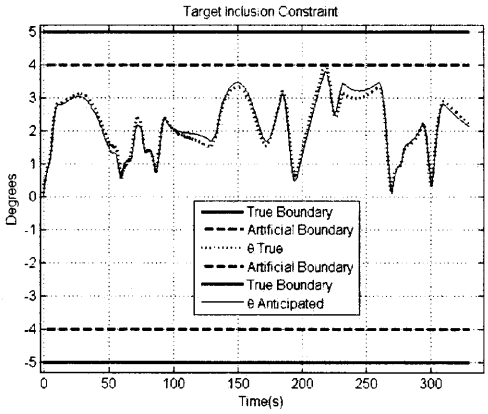

Figure 5-9: Primary camera constraint satisfaction for Scenario 1.

To better illustrate the primary camera constraint, the Figure 5-10 illustrates the trace of the image of the object within the field of view of the primary camera. Again, the dashed line denotes the artificial constraint boundary (CB). The + marks the beginning of the planned trajectory, while the *o* marks the end of the trace. The trace of the image of the object correlates with the angle presented in Figure 5-9. To further illustrate how the solution meets the star camera and communications constraints, Figure 5-11 presents the constraint boundaries in the roll dimension of the trajectory taken by the satellite. This plot is generated in the body frame of the satellite so the constraints move relative to the satellite. Notice that no constraints cross through $\phi = 0$ because this represents the solution that was found to navigate through the constraint boundaries. The angle between $\phi = 0$ to the constraint boundaries should

correlate with Figure 5-8.

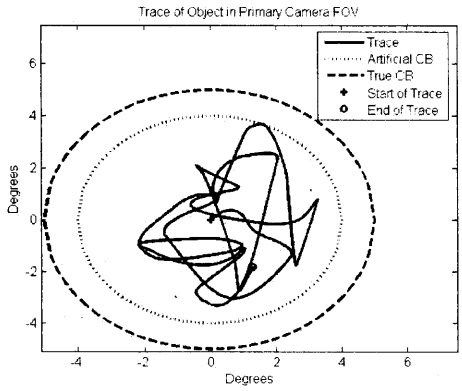

Figure 5-10: Object image trace in the payload camera field of view for Scenario 1.

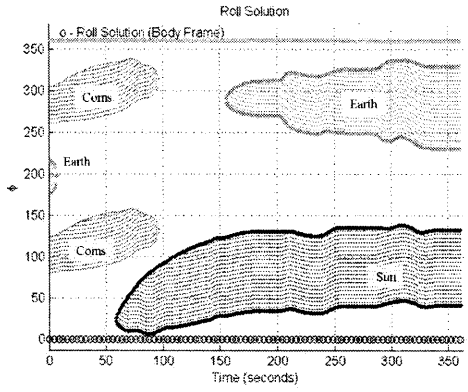

Figure 5-11: Roll solution for Scenario 1.

The true propagation and controller errors are measured throughout the engagement and compared to the expected upper bounds in Figures 5-40 and 5-41. In Figure 5-40 the growth of the propagation error in prediction time is clearly depicted. Each time a new plan is formed the error drops dramatically because each planning segment uses the latest navigation information available. The expected upper limit of the propagation error was successfully determined by the linearized covariance analysis. The controller error reaches its maximum values when the satellite maneuvering with high accelerations. Figure 5-14 provides the angular velocity profile where the large velocity changes correlate with the peaks in the controller error. The results indicate that the Monte Carlo simulation successfully captured the upper limit of the controller error. Finally, for completeness, Figure 5-15 presents the quaternion values throughout the engagement. The jerkiness of the solution is a limitation of the RSTP. The random nature of the motion planner can cause unnecessary wander through the free-space and will likely result in an inefficient trajectory. A smoothing function that works well with randomized trajectory planning algorithms should be developed to improve the performance of the system.

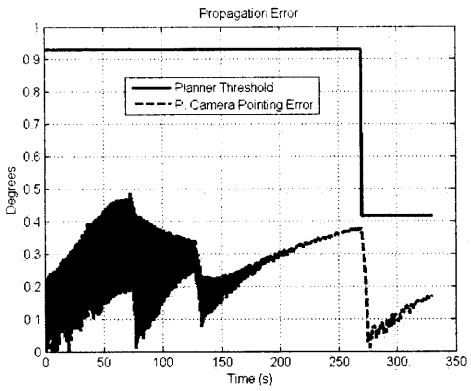

Figure 5-12: The propagation error recorded for Scenario 1.

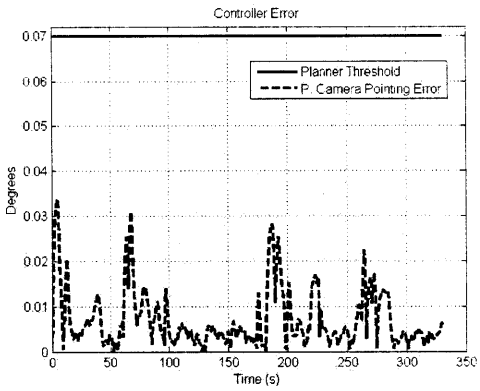

Figure 5-13: The controller error recorded for Scenario 1.

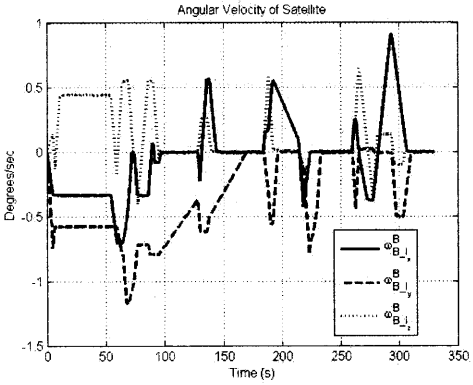

Figure 5-14: The angular velocity profile for Scenario 1.

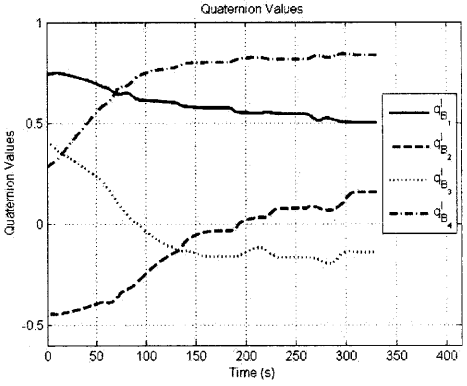

Figure 5-15: The quaternion values for Scenario 1.

5.2 Engagement Scenario 2

Figures 5-16 through 5-29 present the simulation results for Engagement Scenario 2. The plots follow the same formatting presented for Engagement Scenario 1.

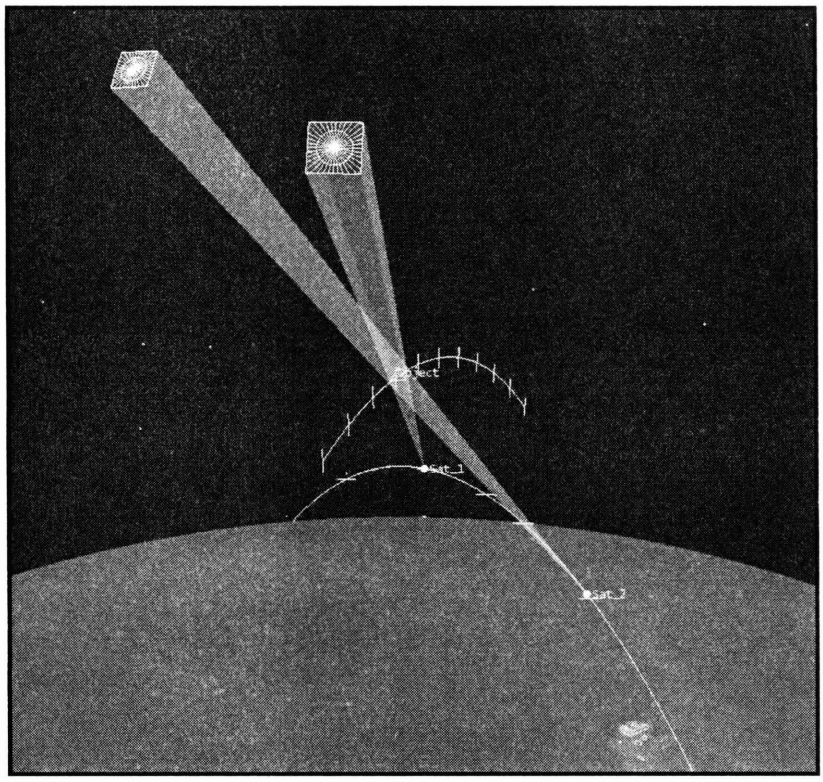

Figure 5-16: Engagement Scenario 2 illustration of orbits.

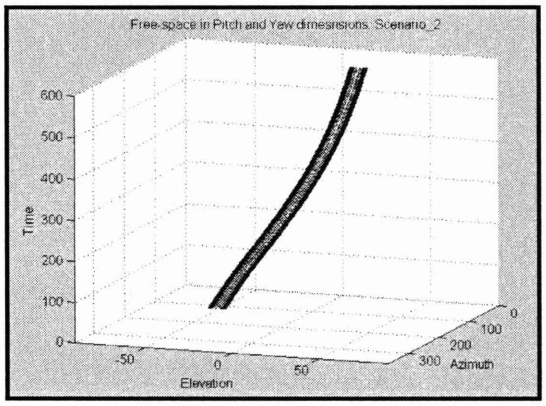

Figure 5-17: Free-space of the pitch and yaw dimensions of engagement Scenario 2.

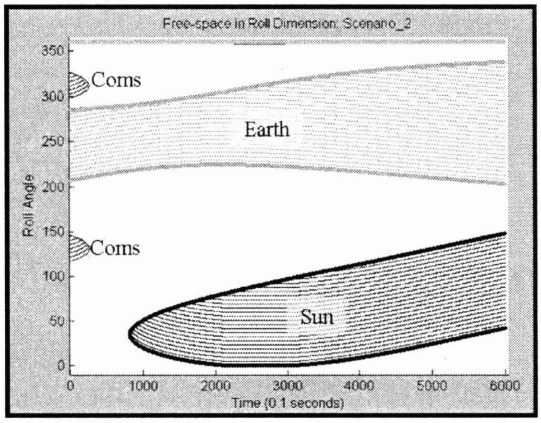

Figure 5-18: Free-space of the roll dimension of engagement Scenario 2.

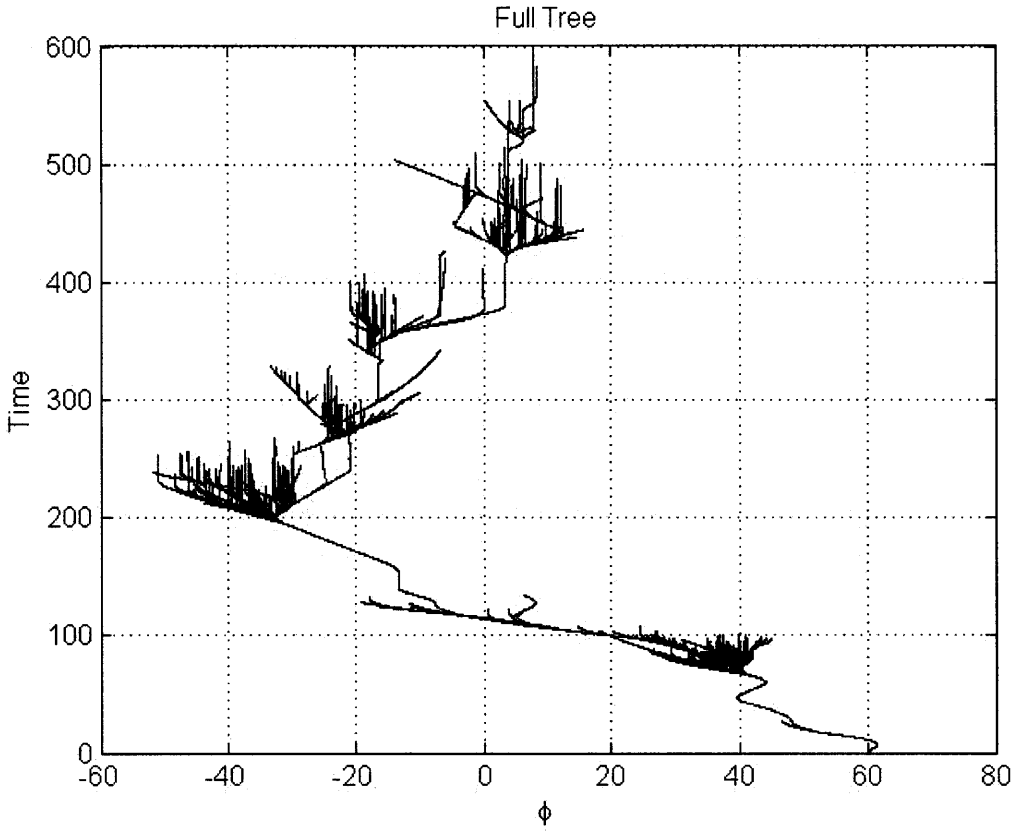

Figure 5-19: Search tree produced in Engagement Scenario 2 viewed in the roll dimension.

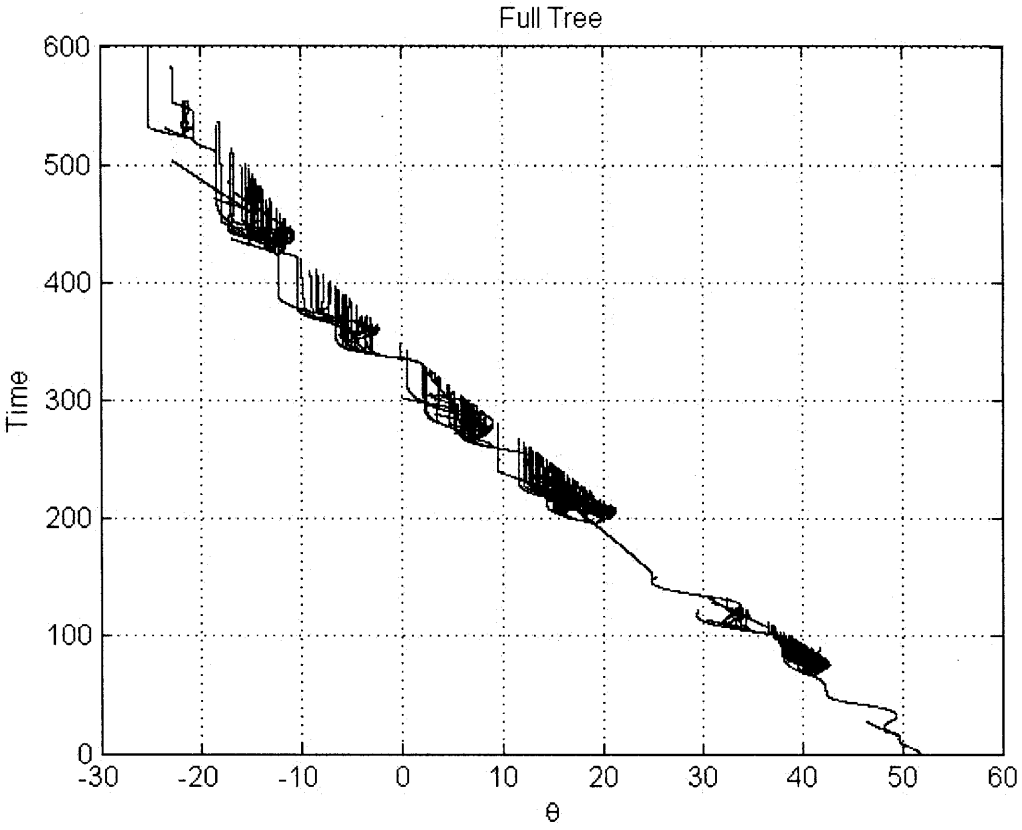

Figure 5-20: Search tree produced in Engagement Scenario 2 viewed in the pitch dimension.

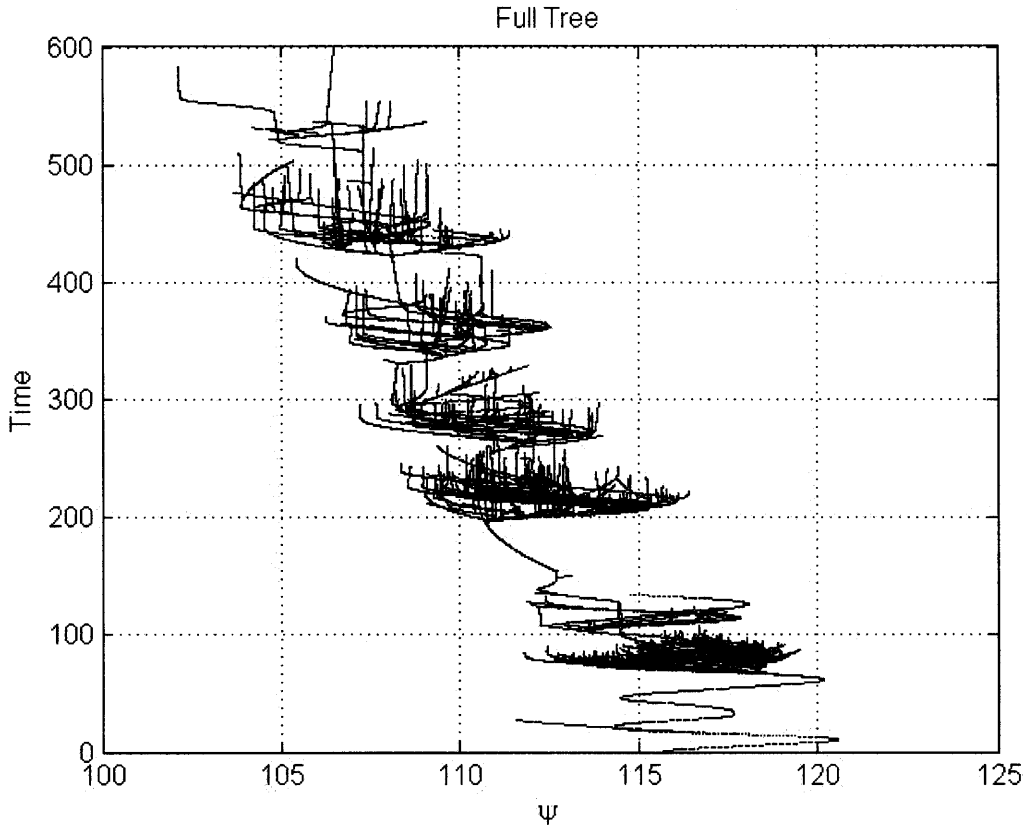

Figure 5-21: Search tree produced in Engagement Scenario 2 viewed in the yaw dimension.

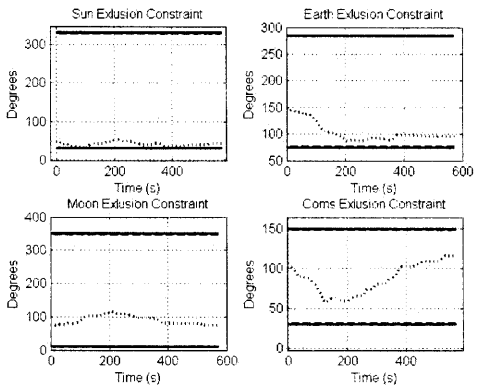

Figure 5-22: Communications and star camera constraint satisfaction for Scenario 2.

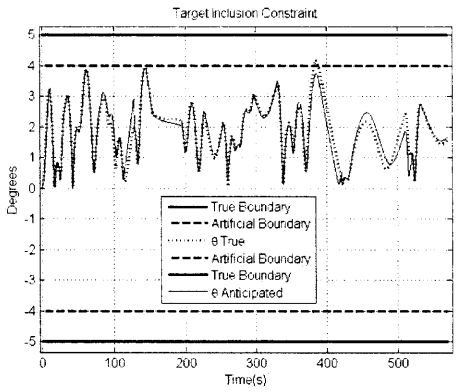

Figure 5-23: Primary camera constraint satisfaction for Scenario 2.

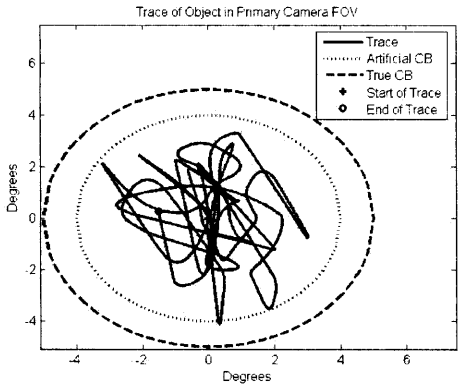

Figure 5-24: Object image trace in the payload camera field of view for Scenario 2.

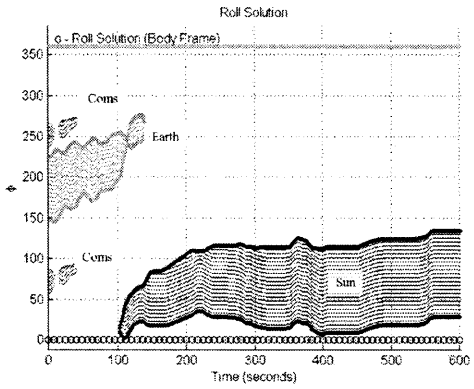

Figure 5-25: Roll solution for Scenario 2.

In Engagement Scenario 2, the upper bound on the expected effect of the propagation error was not chosen conservatively enough. The true effect of the propagation error supercedes the expected value by a few tenths of a degree. The fact that the propagation error exceeded the expected bounds did not result in a constraint violation in this case because the cummulative effect of the propagation and controller errors is less than the expected cummulative effect.

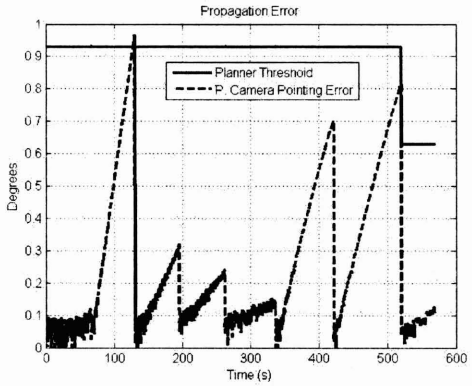

Figure 5-26: The propagation error recorded for Scenario 2.

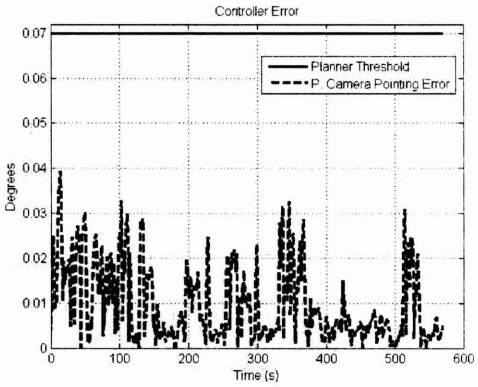

Figure 5-27: The controller error recorded for Scenario 2.

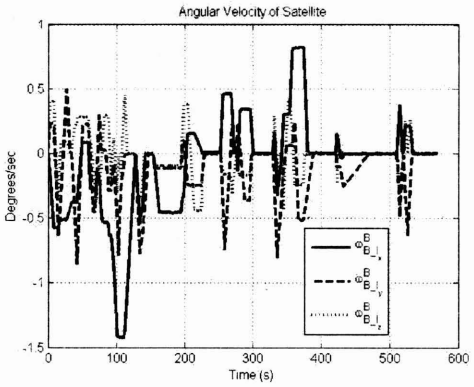

Figure 5-28: The angular velocity profile for Scenario 2.

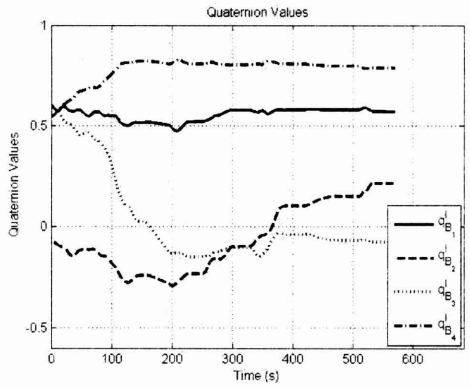

Figure 5-29: The quaternion values for Scenario 2.

5.3 Engagement Scenario 3

Figures 5-30 through 5-43 present the simulation results for Engagement Scenario 3. The plots follow the same formatting presented for Engagement Scenario 1.

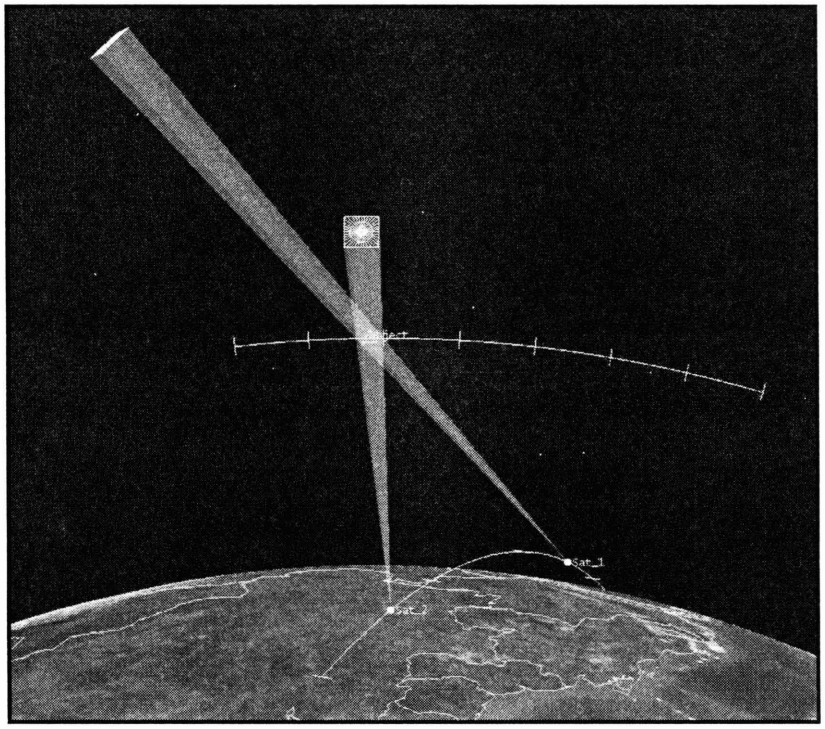

Figure 5-30: Engagement Scenario 3 illustration of orbits.

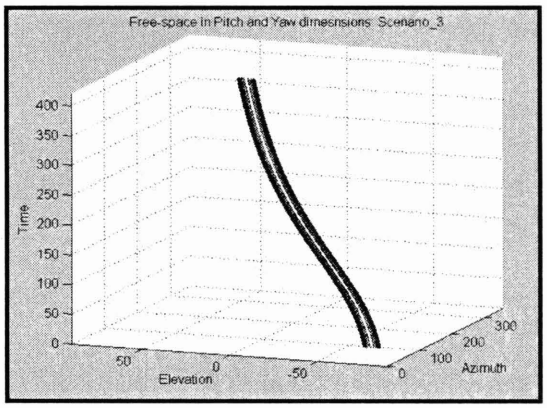

Figure 5-31: Free-space of the pitch and yaw dimensions of engagement Scenario 3.

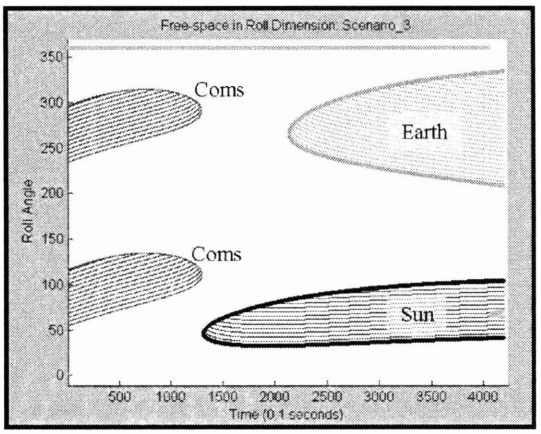

Figure 5-32: Free-space of the roll dimension of engagement Scenario 3.

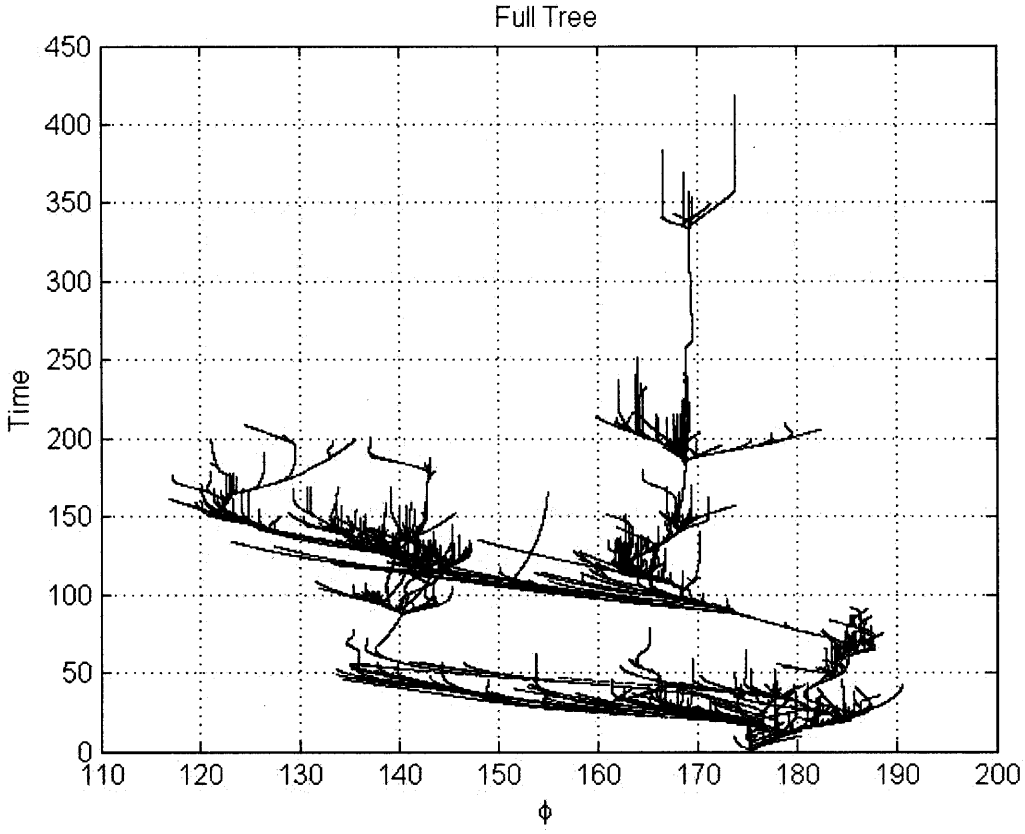

Figure 5-33: Search tree produced in Engagement Scenario 3 viewed in the roll dimension.

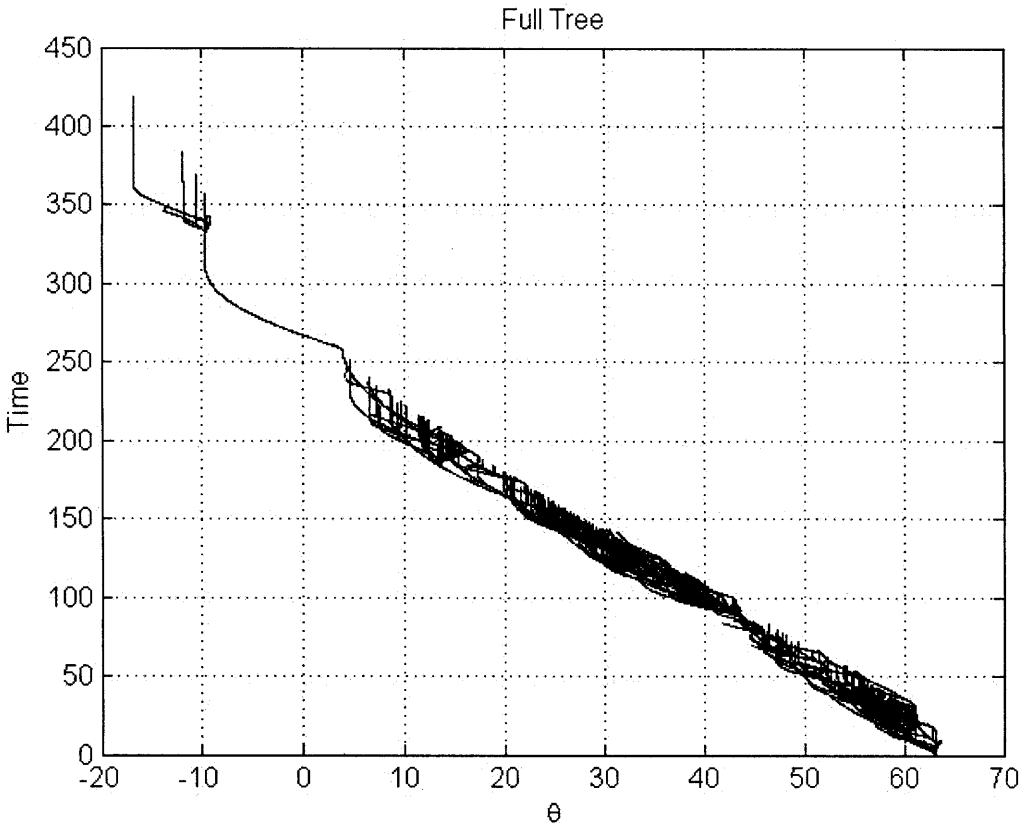

Figure 5-34: Search tree produced in Engagement Scenario 3 viewed in the pitch dimension.

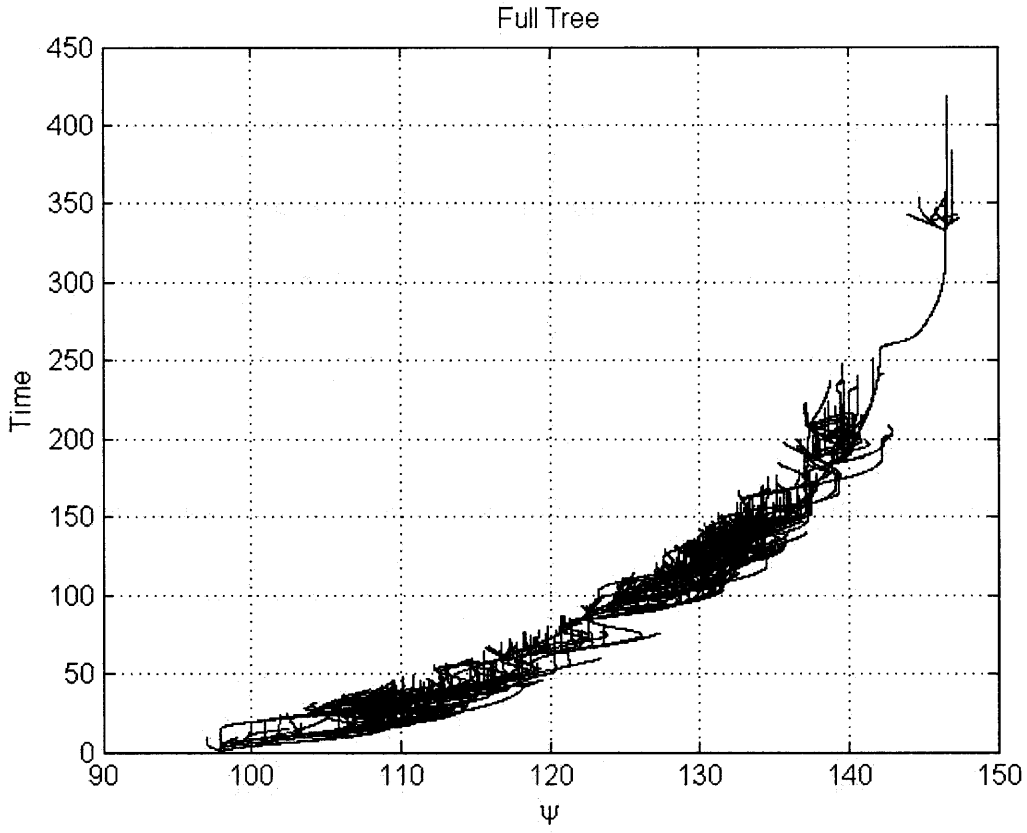

Figure 5-35: Search tree produced in Engagement Scenario 3 viewed in the yaw dimension.

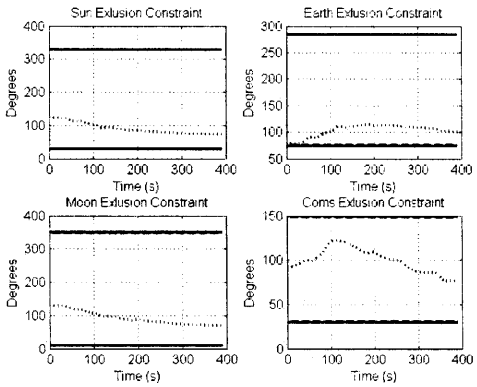

Figure 5-36: Communications and star camera constraint satisfaction for Scenario 3.

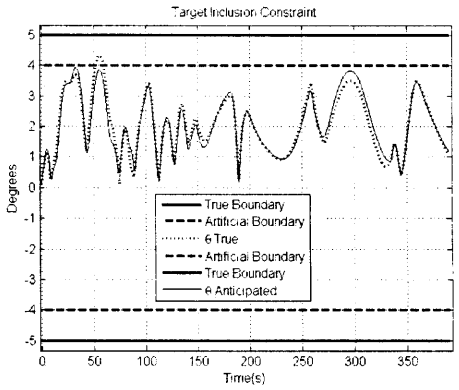

Figure 5-37: Primary camera constraint satisfaction for Scenario 3.

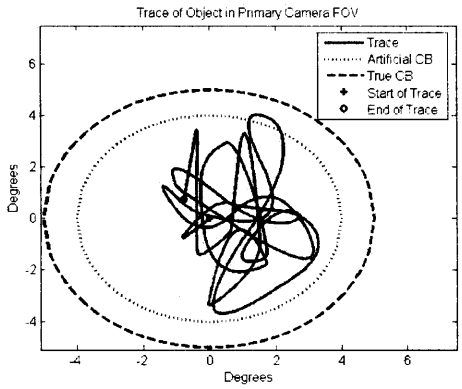

Figure 5-38: Object image trace in the payload camera field of view for Scenario 3.

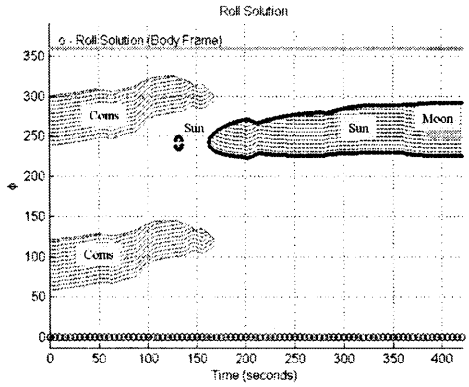

Figure 5-39: Roll solution for Scenario 3.

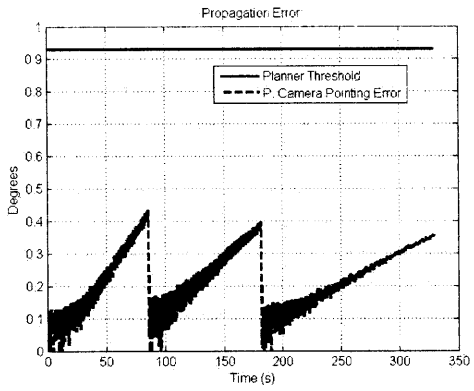

Figure 5-40: The propagation error recorded for Scenario 3.

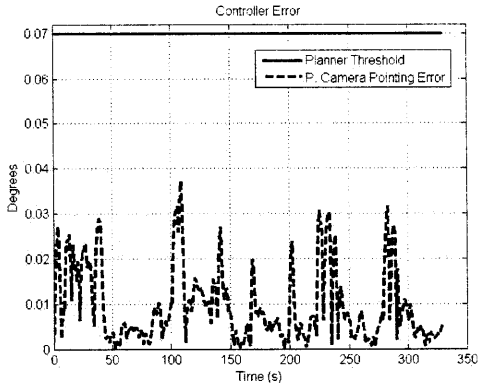

Figure 5-41: The controller error recorded for Scenario 3.

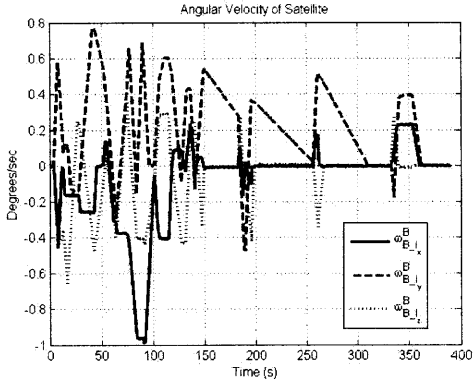

Figure 5-42: The angular velocity profile for Scenario 3.

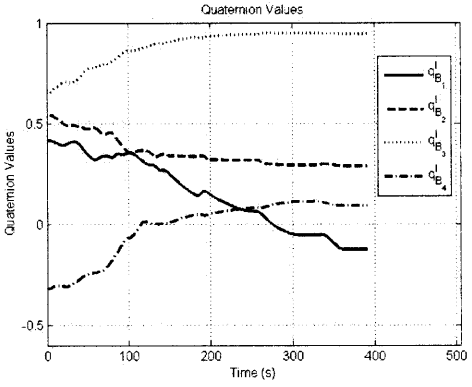

Figure 5-43: The quaternion values for Scenario 3.

5.4 Performance Comparisons

A reproduction of Frazzoli's algorithm is used in an attempt to produce an open-loop solution to Engagement Scenario 1. The configuration space was defined to be the attitude of the satellite and no heuristics were used to alter the algorithm for an improvement in performance. The attempt to produce an open-loop solution with Frazzoli's algorithm was terminated early because the convergence proved to be extremely slow for this problem. After cycling though over 200,000 randomly selected configurations the tree contained a total of 25 nodes and spanned a total of 20.4 seconds. The tightly constrained nature of the problem causes Frazzoli's algorithm to converge extremely slowly. Without altering the algorithm significantly, it cannot be relied upon to produce timely results for such a tightly constrained problem. Table 5.1 compares the number of nodes, unsuccessful attempts to expand the tree, and the average trajectory-segment time span for each of the three Engagement Scenarios using the RSTP and the attempt at solving Engagement Scenario 1 open-loop with Frazzoli's algorithm.

Table 5.1: Planner Statistics

Scenario	**Number of Nodes**	**Number of Dropped Events**	**Average Planning Horizon Span (s)**
1	4082	908	72
2	9218	1178	74.1
3	12858	1769	105
Frazzoli's Algorithm : Open-loop			Total Plan Time (s)
1	25	209,605	20.4

Finally, to illustrate that the RRHC is necessary for providing results that will not violate constraint boundaries, a motion plan is generated for Engagement Scenario 1 and implemented open-loop. The constraint boundaries were artificially enlarged by 1 degree to introduce a level of conservatism into the open-loop solution. Even with the added conservatism, the open-loop solution violates a true constraint boundary. Figure 5-44 illustrates the primary camera constraint. The violation occurs at approximately 82 seconds into the engagement. Figure 5-45 shows how the image of the object strays from the field of view of the primary camera. Figure 5-46 illustrates the unchecked growth of the propagation error, which ultimately causes the constraint violation.

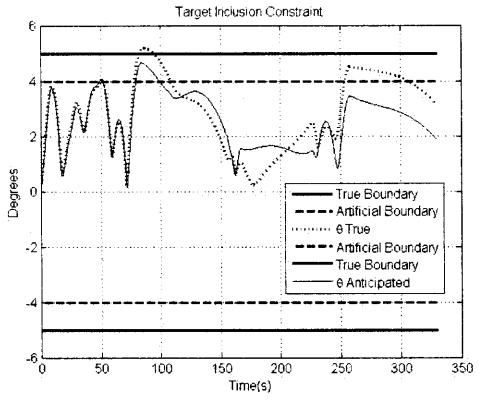

Figure 5-44: Primary camera constraint for open-loop solution of Scenario 1.

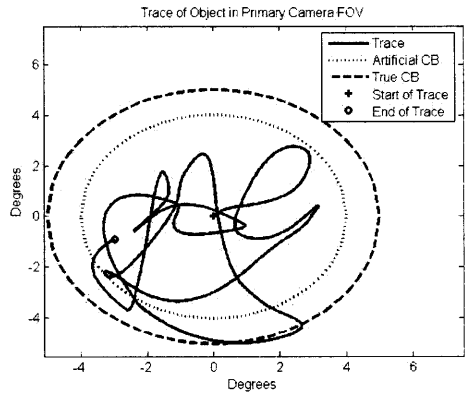

Figure 5-45: Object image trace for open-loop solution to Scenario 1.

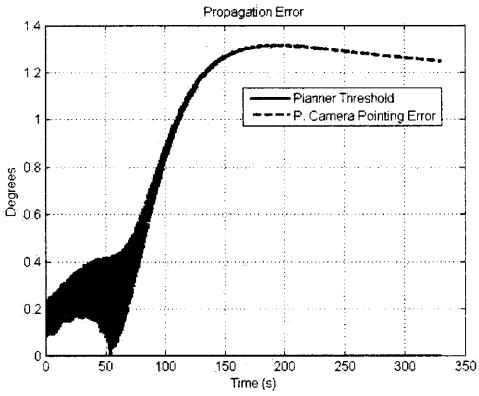

Figure 5-46: Propagation Error for open-loop solution to Scenario 1.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

By closing the guidance, navigation, and control loop around a randomized trajectory planning algorithm, a robotic vehicle can autonomously maneuver through a field of moving obstacles. The guidance strategy provides executable plans that are robust to known error sources when supplied with an estimate of its initial state, a goal, the predicted locations of obstacles, and bounds on error sources affecting the execution of a planned trajectory. The preplanning function described within the RRHC, allows the motion planning solutions to be robust to errors and uncertainties. The preplanning function must be carefully constructed as it determines the amount to artificially enlarge obstacle boundaries within the planning function. If the boundaries be enlarged too much, the free-space is greatly diminished and the randomized trajectory planner suffers in performance; however, if the boundaries are enlarged too little, the plan may not extend far enough into the future to allow for the necessary computation time.

The stability of the RRHC is heavily influenced by three main factors. One, the motion planning segment must extend further into the future than the computation time required to generate the next plan. Two, the initial state of each planning segment must be feasible according to the artificially enlarged constraints, and must have a reachable space that permits search tree expansion. This condition is especially important when revising planned trajectory segments in order to improve the performance of the system (i.e. using a smoothing function), or when updating the constraint boundary definitions. Three, accurate estimates of the maximal effect of system errors on the formulation and execution of a plan are required to establish a balanced tradeoff between constraint boundary expansion and the available free-space. If the effect of errors is underestimated, a constraint violation could result; on the other hand, if the effect of the errors is overestimated, valuable free-space is surrendered to the artificial expansion of constraint boundary definitions.

The RSTP uniformly explores the configuration-$\times$-time space while promoting search tree expansion through the uniformly random selection of baseline maneuvers. Choosing random events (points in configuration-$\times$-time) instead of choosing random

points in the configurations space, allows the logic behind the RRT algorithm to be extended to dynamic environments. Using randomly selected baseline maneuvers as a node-selection metric for search tree expansion promotes the uniform exploration of the configuration-×-time-×-maneuver space, which is a strong argument for the probabilistic completeness of the RSTP algorithm.

The conceptual aspects of the RSTP may be relatively simple, while the specific application may prove somewhat challenging. Decomposing the baseline maneuvers used in search-tree expansion into each dimension separately can greatly simplify the application of the RSTP algorithm whenever such a dimensional decomposition is possible.

6.2 Future Work

Scalability of algorithm: The RRHC and the RSTP algorithm were both designed to be able to solve large scale problems. However, the application of either the RRHC or the RSTP to a high dimensional problem remains to be tested. A possible follow on experiment to test the scalability of both the RRHC and the RSTP would be to apply both to the coordination of multiple satellites tasked with tracking multiple objects.

Application of RSTP to broader spectrum of problems: The RSTP works well and is straight forward to implement for the experiment presented in this thesis. The two main reasons for the ease of implementation for this experiment include the fact that the inner-loop controller provided the control necessary to negate gyroscopic forces and the fact that the satellite was fully actuated, allowing each dimension of the problem to essentially become uncoupled from the others. This certainly will not be the case in general. The development of baseline maneuver sets which fit the requirements of the RSTP may prove more challenging for cases where the problem cannot be decomposed as easily. The development of baseline maneuvers for under-actuated or coupled systems is another area that could use more investigation.

Smoother: For many complex, high dimensional path planning problems, it is enough to find a timely, feasible trajectory to follow. However it is always desirable, and sometimes necessary to have a motion plan that not only avoids obstacles and remains within the dynamic constraints of the vehicle, but also minimizes a performance cost function. The improvement of the trajectory produced by a randomized motion planner lies in the implementation of a smoothing algorithm. Linear or quadratic programming techniques are handy for generating timely results; however, they often require a simplification of the problem formulation (i.e. linearization of dynamics) which results in errors that may destabilize the system. It would be desirable to formulate a smoothing algorithm that works within the mechanisms of the randomized trajectory planner, so timely results can be generated without the need to simplify the problem formulation.

One very rough idea involves iteratively improving upon the trajectory once an initial feasible solution is found. Suppose a performance cost value could be associated with each node. A simple linear search could identify local minima in the cost across an initially feasible trajectory. Connections between nodes representing local minima might be attempted to produce new segments with lower cost than the original segments. If successful, an iterative process may draw the initial trajectory closer to a locally optimal trajectory. Furthermore, because the connections would be produced by the machinery of the planning algorithm, no simplifications to the problem formulation would be necessary.

Appendix A

Notation and Convention

Name	Depiction	Description
Vector	$\mathbf{x}^{\text{frame}}_{\text{component}}$	Bold, lowercase or lowercase greek letter. Reserved for error-free value when estimates are involved. In equations involving more than one reference frame, the frame the vector is constructed in is denoted in the superscript, while any reference to specific components will be in the subscript.
	$\mathbf{x}_k = \mathbf{x}(t_k)$	In equations involving values at discrete times t_k, the time idnex, k, will be denoted in teh subscript.
Matrix	$\mathbf{F}$	Bold, uppercase
Estimate	$\hat{\mathbf{x}}$	Error prone values denoted with a carat
Corrected value	$\bar{\mathbf{x}}$	Denoted with a solid bar
Transformation matrix (DCM)	$\mathbf{T}^{\text{to}}_{\text{from}}$	Bold, uppercase. The orthonormal Direction Cosine Matrix (DCM) rotates from the frame in the subscript to the frame in the superscript.
Quaternion	$\mathbf{q}^{\text{to}}_{\text{from}}$	Bold, lowecase q. Rotates from the frame in the subscript to the frame in the superscript. Represented as column vector where 1^{st} component is scalar.
Transpose	$\mathbf{A}'$	Prime
Inverse	$\mathbf{A}^{-1}$	Superscript (-1)

Name	Depiction	Description
Skew symmetric matrix	$[\mathbf{x}\times] = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}$	The skew symmetric matrix can be used in place of the cross product operation, i.e. $\mathbf{x} \times \mathbf{y} = [\mathbf{x}\times]\mathbf{y}$
Angular velocity	$\omega^C_{A\_B}$	Denotes the angular velocity of frame A with respect to frame B, coordinatized in frame C.
Set	$\mathcal{X}$	Scripted capital letter
Function	F	Italisized capital letter
Euclidian norm	$\|\|\mathbf{x}\|\|$	Two vertical bars on either side of the variable
Quaternion multiplication	$\mathbf{q}^C_A = \mathbf{q}^B_A \otimes \mathbf{q}^C_B$ $\mathbf{q}\otimes = \begin{bmatrix} q_1 & -q_2 & -q_3 & -q_4 \\ q_2 & q_1 & -q_4 & q_3 \\ q_3 & q_4 & q_1 & -q_2 \\ q4 & -q_3 & q_2 & q_1 \end{bmatrix}$ $\mathbf{q}^C_A = [\mathbf{q}^C_B\otimes_{(-)}]\mathbf{q}^B_A$ $[\mathbf{q}\otimes_{(-)}] = \begin{bmatrix} q_1 & -q_2 & -q_3 & -q_4 \\ q_2 & q_1 & q_4 & -q_3 \\ q_3 & -q_4 & q_1 & q_2 \\ q4 & q_3 & -q_2 & q_1 \end{bmatrix}$	Encircled cross product sign.
Frames	B I SC PC	Body Inertial Star Camera Primary Camera

Bibliography

[1] Anna Atramentov and Steven LaValle. Efficient nearest neighbor searching for motion planning. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, Washington, DC, May 2002.

[2] Vernon Barger and Martin Olsson. *Classical Mechanics: A Modern Perspective.* McGraw-Hill, Inc., New York, NY, 1995.

[3] Richard Battin. *An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition.* American Institute of Aeronautics and Astronautics, Inc., Reston, VA, 1999.

[4] Dimitri Bertsekas. *Nonlinear Programming.* Athena Scentific, Belmont, MA, 1999.

[5] John Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–205, March 1998.

[6] Vaerie Boor, Mark Overmars, and A. Frank van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, Detroit, Michigan, May 1999.

[7] L. Breger, P. Ferguson, J. How, S. Thomas, T. McLaughlin, and M. Campbell. Distributed control of formation flying spacecraft built on OA. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, Texas, August 2003.

[8] L. Breger and J. How. Formation flying control for the MMS mission using GVE-Based MPC. In *Proceedings of the 2005 IEEE Conference on Control Applications*, pages 565–570, Toronto, Canada, August 2003.

[9] L. Breger and J. How. J_2 - modified GVE-Based MPC for formation flying spacecraft. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, California, August 2005.

[10] Louis Breger. Model predictive control for formation flying spacecraft. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautical and Astronautical Engineering, June 2004.

[11] J. F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, Massachusetts, 1988.

[12] Peng Cheng and Steven LaValle. Reducing metric sensitivity in randomized trajectory design. In *Proceedings of the 2001 IEEE/RSJ*, International Conference on Intelligent Robots and Systems, pages 46–48, Maui, HI, October 2001. IEEE.

[13] C. Clark and S. Rock. Randomized motion planning for groups of nonholonomic robots, 2001.

[14] Paul DeRusso, Rob Roy, Charles Close, and Alan Desrochers. *State Variables for Engineers.* John Wiley and Sons, Inc, New York, NY, 1998.

[15] E. Frazzoli, M. Dahleh, and E. Feron R. Kornfeld. A randomized attitude slew planning algorithm for autonomous spacecraft. In *AIAA Guidance, Navigation, and Control Conference*, number 2001-4155, Montreal, Canada, August 2001. AIAA.

[16] Emilio Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning.* PhD dissertation, MIT, Department of Aeronautics and Astronautics, June 2001.

[17] Emilio Frazzoli, Munther Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, January 2002.

[18] Ian Garcia. Nonlinear trajectory optimization with path constraints applied to spacecraft reconfiguration maneuvers. Master's thesis, Massachusetts Institute of Technology, 2005.

[19] Arthur Gelb. *Applied Optimal Estimation.* MIT Press, Cambridge, Massachusetts, 1974.

[20] E. Gillies, A. Johnston, and C. McInnes. Action selection algorithms for autonomous microspacecraft. *Journal of Guidance*, 22(6):914–916, June 1999.

[21] David Hsu, Tingting Jiang, John Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, 2003.

[22] David Hsu, Robert Kindel, Jean-Clead Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. In *Proc. Workshop on Algorithmic Foundations of Robotics (WAFR '00)*, Hanover, NH, March 2000.

[23] Piort Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. Technical report, Stanford University, 1999.

[24] A. Jadbabaie, J. Yu, and J. Hauser. Stabilizing receding horizon control of nonlinear systems: A control Lyapunov function approach, 1999.

[25] Leonard Jaillet and Thierry Simeon. A PRM-based motion planner for dynamically changing environments. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 2004. IEEE.

[26] James Kuffner Jr. and Steven LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the 2000 International Conference on Robotics and Automation*, San Francisco, California, April 2000. IEEE.

[27] Wei Kang and Andrew Sparks. Coordinated attitude and formation control of multi-satellite systems. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA 2002-4655, Monterey, CA, August 2002. AIAA.

[28] Lydia Kavraki, Mihail Kolountzakis, and Jean-Claude Latombe. Analysis of probablistic roadmaps for path planning. In *Proc. 1996 International Conference on Robotics and Automation*, Minneapolis, Minnesota, April 1996. IEEE.

[29] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12(4):566–580, 1996.

[30] J. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 1999.

[31] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, Massachusetts, 1991.

[32] S. LaValle, H. nos, C. Becker, and J. Latombe. Motion strategies for maintaining visibility of a moving target, 1997.

[33] Steven LaValle and James Kuffner Jr. Rapidly-exploring Random Trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics*, 2000.

[34] Steven LaValle, Zoujun Shen, and Peng Cheng. RRT-based trajectory design for autonomous automobiles and spacecraft. *Archives of Control Sciences*, 11(3-4):51–79, 2001.

[35] Frank Lewis. *Optimal Control.* John Wiley and Sons, Inc., New York, NY, 1986.

[36] Rami Mangoubi. *Robust Estimation and Failure Detection: A Concise Treatment.* Springer, New York, NY, 1998.

[37] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, November 1999.

[38] Marc McConley, Brent Appleby, Munther Dahleh, and Eric Feron. A computationally efficient Lyapunov-based scheduling procedure for control of nonlinear systems with stability guarantees. *IEEE Transactions on Automatic Control*, 45(1):33–49, January 2000.

[39] Mark Milam, Kudah Mushambi, and Richard Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *2000 Conference on Decision and Control*, Pasadena, CA, 2000.

[40] M. Mitchell, L. Breger, and J. How. Effects of navigation filter properties on formation flying control. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island, August 2004.

[41] Richard M. Murray, Li Zexiang, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, New York, New York, 1994.

[42] John H. Reif and Micha Sharir. Motion planning in the presence of moving obstacles. In *IEEE Symposium on Foundations of Computer Science*, pages 144–154, 1985.

[43] Arthur Richards. Trajectory optimization using mixed-integer linear programming. Master's thesis, Massachusetts Institute of Technology, 2002.

[44] Mitul Saha, Jean-Claude Latombe, Yu-Chi Chang, and Friedrich Prinz. Finding narrow passages with probabilistic roadmaps: The small-step retraction method. *Autonomous Robots*, 19(3):301–319, 2005.

[45] Bernard Schutz. *A First Course in General Relativity.* Cambridge University Press, Cambridge, United Kingdom, 1985.

[46] P. Sedgewick. *Algorithms.* Addison-Wesley, second edition, 1988.

[47] P. Stadter, G. Barrett, D. Watson, T. Esposito, and J. Bristow. Autonomous command and control for distributed spacecraft systems. In *NanoTech 2002*, number 2002-5725, Houston, TX, September 2002. AIAA.

[48] Edwin Taylor and John Wheeler. *Spacetime Physics.* W.H. Freeman and Company, San Francisco, CA, 1966.

[49] M. Tillerson, L. Breger, and J. How. Distributed coordination and control of formation flying spacecraft. In *Proceedings of the 2003 American Control Conference*, volume 2, pages 1740–1745, 2003.

[50] Chris Urmson and Reid Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Las Vegas, Navada, October 2003.

[51] Prahlad Vadakkepat, Kay Chen, and Wang Ming-Liang. Evolutionary artificial potential fields and their application in real time robot path planning. Report, University of Singapore, Department of Electrical Engineering, University of Singapore.

[52] James Wertz. *Spacecraft Attitude Determination and Control.* D. Reidel Publishing Company, Boston, Massachusetts, 1986.

[53] Bong Wie, David Baily, and Christopher Heiberg. Rapid multi-target acquisition and pointing control of agile spacecraft. In *AIAA Guidance, Navigation, and Control Conference*, number A00-37187, Denver, CO, August 2000. AIAA.